Trabajo de Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Programación de algoritmos de identificación de anuncios en tiempo real

Autor: Miguel Salcedo Trigo
Tutor: José Ramón Cerquides Bueno

Dpto. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Trabajo de Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Programación de algoritmos de identificación de anuncios en tiempo real

Autor:
Miguel Salcedo Trigo

Tutor:
José Ramón Cerquides Bueno
Profesor titular

Dpto. de Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2021

Trabajo de Fin de Grado: Programación de algoritmos de identificación de anuncios en tiempo real

Autor: Miguel Salcedo Trigo

Tutor: José Ramón Cerquides Bueno

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2021

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

Ha sido un año difícil y complicado, marcado por la magnitud de una pandemia, la del Covid 19, en la que el teletrabajo se ha convertido en la alternativa a la que todos hemos tenido que adaptarnos de forma precipitada.

En primer lugar, me gustaría mostrar mi más sincero agradecimiento a mi tutor, D. José Ramón Cerquides Bueno, por ayudarme, guiarme y orientarme en todo este proceso de tutorización de Fin de Grado, por todas las sugerencias para mejorar mi trabajo, por haber minimizado las dificultades que presenta un trabajo a distancia, con una comunicación fluida, rápida y certera en todo momento.

A la Universidad de Sevilla, por facilitar los medios necesarios para continuar con los estudios de forma telemática.

Agradezco también, a todos los profesores que he tenido en la ETSI, porque desde que comencé mis estudios hasta ahora han sido los grandes pilares que han contribuido a mi formación y gracias a ellos me han confirmado que fue una buena decisión la de optar por estos estudios.

Y por último y no menos importante, a mi familia, por estar siempre ahí, por su constante apoyo y confianza, mi fuente de motivación en este recorrido, y como no, a mis amigos, por acompañarme en esta etapa y hacer mi paso por la Universidad de Sevilla mucho más agradable.

Gracias a todos.

Miguel Salcedo Trigo

Sevilla, 2021

Resumen

El incremento en el consumo de contenido audiovisual se ha vuelto más que notable en los últimos años. Hoy en día existe una enorme cantidad de vídeos que están disponibles para ver en cualquier momento, ya sea en directo o no, y en cualquier plataforma (televisión, Internet o aplicaciones para teléfonos inteligentes).

Esto ha provocado el aumento del interés por los sistemas de Video Fingerprinting, que son utilizados para extraer un identificador del vídeo, denominado huella o *hash*, utilizado para poder comparar los vídeos entre sí. Cada vez son más las técnicas propuestas para calcular las huellas de los vídeos y su elección para un sistema de Video Fingerprinting dependerá del uso final de este sistema.

Este documento recoge, por un lado, el proceso de actualización de una aplicación que implementa un sistema de Video Fingerprinting para la detección de anuncios de televisión a tiempo real, adaptando la aplicación a las librerías más modernas utilizadas para la ingesta y manipulación del *streaming*.

Por otro lado, presenta las tareas de búsqueda y elección de un nuevo método para el sistema de Video Fingerprinting de la aplicación, su posterior implementación en código fuente y las pruebas y comparaciones entre el nuevo método propuesto y el implementado anteriormente en la aplicación.

Abstract

The increase in the consumption of audiovisual content has become more than notable during the last years. Nowadays there is a big quantity of videos which are available at any time, they being live or not, and in any platform (tv, Internet or smartphone apps).

This has provoked the rise in the interest for video fingerprinting systems, which are used to extract an identifier of the video called fingerprint, which makes the comparison of videos possible. More and more techniques to calculate fingerprints are proposed and its election for its video fingerprinting system will depends on the final use of the system.

This document gathers, one the one hand, the update process of an application which implements a video fingerprinting system for the detection of TV adverts in real time, adapting the application to the most modern libraries, used for the ingestion and manipulation of the streaming.

On the other hand, it presents the searching tasks and election of a new method for the video fingerprinting system of the application, its subsequent implementation in source code and the checking and comparative between the new proposed method against the one used by the application previously.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

ÍNDICE DE GRÁFICOS

1 INTRODUCCIÓN

Inspiration exists,
but it has to find you working.
- Pablo Picasso -

El aumento en la distribución y visualización de la información multimedia a través de cualquier plataforma y medio como puede ser la televisión, Internet, aplicaciones para Smartphones, etc., ha atraído una considerable atención hacia los sistemas de Video Hashing o Vídeo Fingerprinting. Estos sistemas se encargan de obtener una huella única para cada vídeo, como si de una etiqueta se tratara, para poder comparar los vídeos entre sí.

Según el estudio [1] realizado por Barlovento Comunicación en julio de 2021, que integra los datos de consumo de contenidos audiovisuales por televisión, ordenador y móvil, los individuos mayores de 18 años de edad consumieron un promedio de 273 minutos al día de contenidos audiovisuales. Y, aunque el aumento en el consumo de nuevas plataformas de streaming como Netflix, Youtube o Twitch es innegable, especialmente en jóvenes menores de 25 años, la televisión tradicional sigue siendo el medio más utilizado para la visualización de contenido audiovisual con un 73,5% del total del consumo audiovisual.

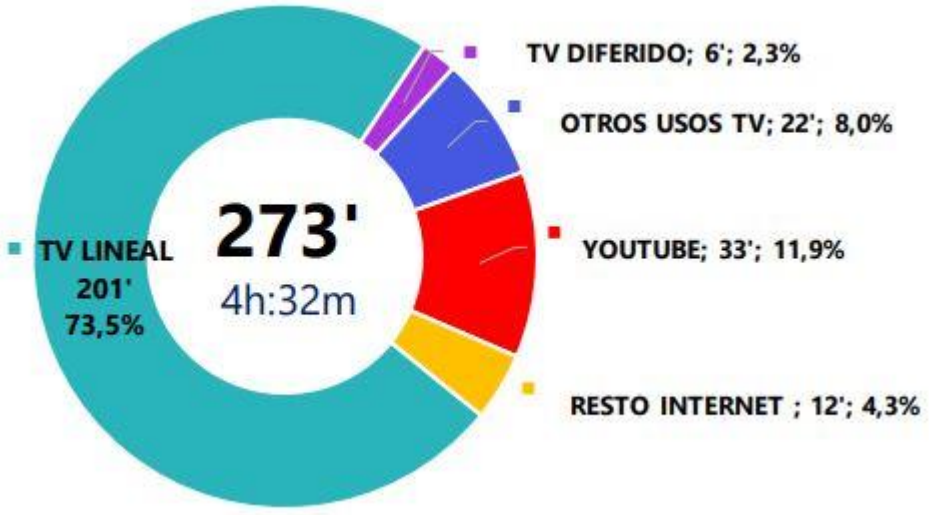

Figura 1. *Share* (cuota) audiovisual agregado - Julio 2021.

El porcentaje tan elevado del consumo de televisión frente a otras plataformas, hace que las marcas sigan apostando por la televisión como medio preferido para emitir sus spots publicitarios, uno de los recursos más eficaces para anunciarse. Y es que, gracias a las técnicas narrativas, como los estereotipos, arquetipos y la música, se puede enviar una gran cantidad de información en poco tiempo.

Por todos estos motivos, el Dpto. de Teoría de la Señal y Comunicaciones de la Universidad de Sevilla desarrolló un software llamado Sistema Inteligente de Reconocimiento de Anuncios de Televisión (SIRA-TV). Este programa implementa un sistema de Vídeo Fingerprinting, que, aplicado en un streaming de televisión, sirve para detectar y reconocer los anuncios a tiempo real.
Con el paso de los años el incremento en la calidad del material audiovisual se ha hecho más que notable, y a día de hoy se siguen buscando nuevas formas de que el contenido que llegue al espectador sea lo más parecido posible a la realidad. Este motivo impulsa a seguir actualizando y desarrollando SIRA-TV para que el software pueda trabajar con los estándares, formatos, resoluciones, librerías y funciones más modernas.

En los Capítulos 2, 3 y 4 de este documento, se introducirá al lector en el estado del arte de los sistemas de Video Fingerprinting. También se le hará un breve resumen de la estructura y funcionamiento de la aplicación SIRA-TV, así como del software utilizado. Todo esto para acercar al lector a la aplicación y que pueda comprender con más facilidad las tareas de actualización realizadas que se detallan en el Capítulo 5. A continuación, en los Capítulos 6 y 7 se presenta el algoritmo de hashing elegido para el sistema de Video Fingerprinting de la aplicación y se muestra la batería de pruebas que se utilizó para comprobar la eficacia de los dos algoritmos de hashing. Por último, en el Capítulo 8 se hace una conclusión en base a los resultados y se informa sobre las posibles líneas de futuro.

2 FUNDAMENTOS TEÓRICOS Y ESTADO DEL ARTE

Un sistema de Video Fingerprinting (o video hashing) tiene como objetivo detectar si un vídeo es una copia (alterada o no) de un video de referencia mediante la extracción de características basadas en su contenido: huellas (o hash).

En un origen, se utilizaban métodos de hashing perceptual de imágenes para generar el código hash del vídeo directamente. Es preciso señalar la diferencia entre el hashing de vídeo y el hashing de imágenes. En el hashing de imágenes solo se utiliza la información espacial, no hay información temporal. Es por eso por lo que en numerosos sistemas de Video Fingerprinting, se utiliza hashing perceptual de imágenes sobre unos determinados frames del vídeo, o *keyframes,* para generar la huella del vídeo, sin considerar la estructura temporal del vídeo.

La huella de una imagen, es una cadena binaria de longitud fija que se obtiene tras realizar una serie de operaciones en función a un algoritmo que se aplica sobre las características de dicha imagen.

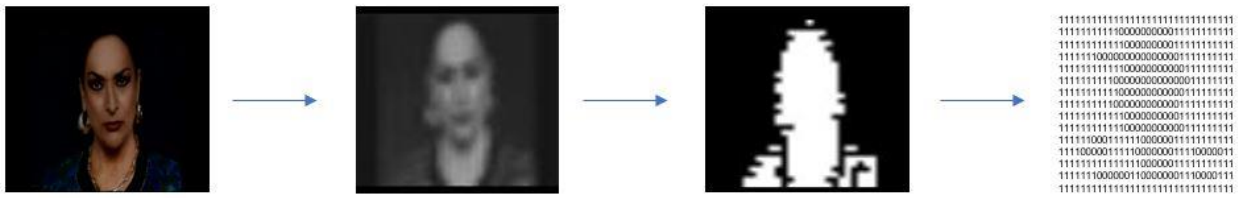

Figura 2. Proceso de Perceptual Hashing aplicado al frame de un anuncio publicitario.

Los sistemas de Video Fingerprinting se componen de dos etapas diferentes: online y offline. En la etapa offline, las huellas de los vídeos originales o de referencia se extraen y se guardan en una base de datos (ver Figura 3). En la etapa online, la huella se extrae del vídeo que se está consultando en el momento y a continuación, se busca y se compara con las de los vídeos de la base de datos. (ver Figura 4).

Por ejemplo, en [2] se divide un sistema de Video Fingerprinting en tres componentes esenciales: una base de datos de huellas, la extracción de las huellas y la identificación de las huellas. La base de datos guarda las huellas con su correspondiente información, como puede ser el nombre del contenido o del productor. La extracción de la huella se puede clasificar en función del tipo de características del contenido que se utilicen para calcular la huella. Y por último, la búsqueda e identificación se aplican para obtener coincidencias entre las huellas con la base de datos.

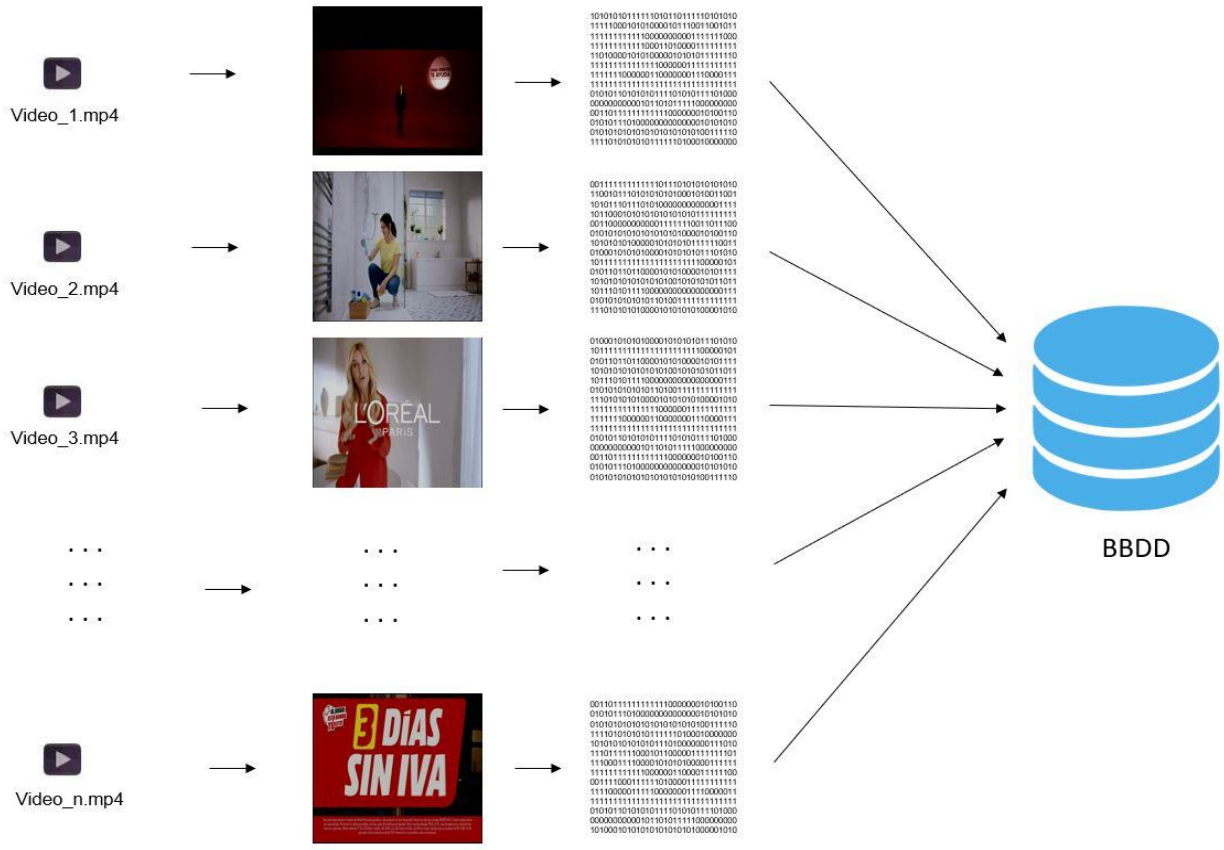

Figura 3. Etapa offline en un sistema de Video Fingerprinting.

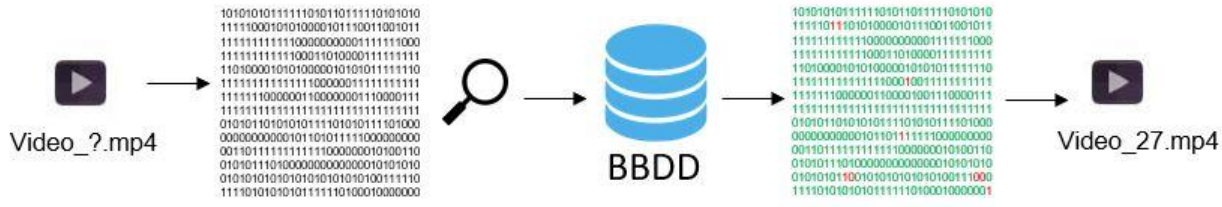

Figura 4. Etapa online en un sistema de Video Fingerprinting.

Actualmente, debido al gran aumento en el número de vídeos disponibles en Internet y a la capacidad de almacenamiento de las grandes bases de datos que contienen dichos vídeos, estamos observando un cambio en los sistemas de Video Fingerprinting, una nueva tendencia, como proponen los estudios [3] y [4], que le dan un giro a la estrategia utilizada comúnmente por los sistemas, el hashing perceptual de imágenes. Estos sistemas, para extraer la huella, utilizan tanto la información espacial del vídeo como la temporal. Su fundamentación consiste en que un vídeo es más que una sucesión de frames, dado que contiene mucha información temporal.

En [3] se propone un algoritmo para evitar la distribución de vídeos duplicados, que se generan a partir de vídeos existentes infringiendo ilegalmente las normas de copyright, un problema frecuente hoy en día, debido a la facilidad que existe para guardar vídeo, editarlo y distribuirlo en Internet. La utilidad de este algoritmo se potencia cuando es necesaria una búsqueda de vídeos en bases de datos a gran escala.

En [4] se muestra un método que parte de un concepto clave, no todos los frames tienen la misma influencia a la hora de representar el vídeo. Normalmente, solo unos pocos frames tienen la capacidad de representar el vídeo completo, el resto pueden contener ruido y efectos nocivos. El método consta de una red neuronal convolucional (CNN) y una red de gran memoría de corto plazo (LSTM) que reciben la información espacial y temporal de los frames a la vez, y se entrenan para asignarle a los frames diferente peso en función de su importancia, según un algoritmo que denomina algoritmo de atención. Por último, genera el hash del vídeo.

Estos nuevos mecanismos proporcionan una gran fiabilidad y son óptimos cuando se quiere hacer consultas en grandes bases de datos, como las de Internet. Sin embargo, en el caso que aborda este proyecto, el de los mensajes publicitarios, el sistema de Video Fingerprinting estará basado en el hashing perceptual de imágenes a partir de *keyframes*. Esta elección se debe a dos motivos principales: la brevedad en la duración de los spots convencionales de televisión (en el Apartado 2.2.1 se hará hincapié en los estándares de los mensajes publicitarios) y el número reducido de anuncios que contiene la base de datos. Si se quisiera clasificar vídeos de larga duración, o hacer consultas en bases de datos de gran escala, habría que plantearse otros métodos de Video Fingerprinting más adecuados, como los citados anteriormente [3], [4].

2.1 Perceptual Hashing

El hashing perceptual, también llamado *content-based fingerprinting*, cómo se explica en [5] y [6], son métodos que convierten un vídeo en un hash basándose en su contenido. Se utiliza en muchas aplicaciones tales como en la recuperación de vídeos, detección de copias o manipulaciones, análisis forense [7] y en la evaluación de la calidad del vídeo. El hash o huella extraído debe cumplir dos propiedades fundamentales:

- Robustez: significa que dos vídeos con el mismo contenido deben tener un hash aproximadamente igual, a pesar de que hayan sufrido transformaciones que no afecten a su contenido, como la recompresión.

- Discriminabilidad: significa que dos vídeos perceptualmente distintos deben tener huellas significativamente distintas.

Hay diferentes estrategias de hashing perceptual, por ejemplo, en [8] se propone un método que calcula la huella utilizando los límites del cuadro, esto puede ser útil para detectar la copia de una película de larga duración, pero no para el caso de evaluar vídeos de corta duración.

Otra forma de calcular el hash de un vídeo es usar sus descriptores globales como pueden ser el movimiento, el color y la distribución espacial-temporal de la intensidad de los píxeles [9].

Como se ha explicado en el apartado anterior, el objetivo será trabajar con un sistema de Video Fingerprinting que se fundamente en hashing de imágenes. Los estudios realizados en los artículos [10] y [11] nos acercan mucho más al hashing de imágenes y añaden propiedades adicionales que los métodos de hashing perceptual de imágenes deben cumplir:

- Compactabilidad: el valor del hash debe de ser mucho más pequeño que el valor original de la imagen.
- Impredecibilidad: el valor del hash no puede ser calculado sin una clave secreta.

2.2 Detección de anuncios de televisión mediante la aplicación SIRA-TV

Como se ha indicado en los apartados anteriores, el uso de sistemas de Video Fingerprinting es cada vez más común y son muchas las estrategias que se proponen en función de la finalidad del sistema final.

A continuación, se especifica el tipo de sistema implementado en la aplicación SIRA-TV en función del objetivo deseado, la detección de anuncios de televisión a tiempo real.

2.2.1 Propiedades de los anuncios de televisión

Según la legislación audiovisual española [12] los canales de televisión no deben emitir más de 12 minutos de mensajes publicitarios por hora de reloj, otros 5 minutos adicionales para anuncios de sus propios programas, y 3 minutos para telepromoción. Esto deja a los canales con 20 minutos por hora de reloj de programación destinada a anuncios. La duración de un anuncio de televisión oscila entre los 10 y los 70 segundos, estando la media en 20 segundos.

Este dato supone la primera premisa con la que el sistema de Video Fingerprinting de SIRA-TV trabajará: la rapidez en la extracción de la huella.

Las campañas publicitarias de las que salen los anuncios suelen durar semanas o pocos meses. Lo que provoca que la base de datos necesite ser actualizada constantemente y muchos vídeos se van a quedar obsoletos después de cada actualización; así pues, no se necesitará un gran número de vídeos en la base de datos.

Esto nos aleja de los métodos de Video Fingerprinting orientados a hacer consultas en bases de datos con miles y miles de vídeos.

Ante estas condiciones de partida, se eligió un algoritmo de Perceptual Hashing que implementa la Transformada Discreta del Coseno de dos dimensiones (2D-DCT) para el sistema de Video Fingerprinting de la aplicación SIRA-TV.

Este proyecto propone implementar un algoritmo de Perceptual Hashing diferente en la aplicación, llamado *average hashing* o "ahash".

Las comparaciones entre ambos algoritmos están detalladas en los Capítulos 6 y 7.

2.2.2 Propiedades de las huellas y la base de datos

El resultado del sistema de Video Fingerprinting que implementa la aplicación SIRA-TV será una huella de 1024 bits de cada anuncio, realizada sobre dos frames distintos de este, el primer frame no negro del anuncio y un frame de *offset*¸ fijado 25 frames después. La utilidad de este frame es decidir situaciones en las que la búsqueda en la base de datos, utilizando la huella del primer frame no negro, devuelva más de un candidato o ninguno.

Resultaría lógico plantearse el hecho de utilizar más frames de *offset* para detectar inequívocamente los vídeos. Pero, tras realizar las comprobaciones pertinentes, con un solo frame de *offset* se consiguen los resultados óptimos utilizando el menor coste computacional posible. Influye en esto la naturaleza de los anuncios que,

debido a su brevedad, sufren cambios bruscos de secuencias en poco tiempo. Con lo que los frames 1 y 26 diferirán lo suficiente para proporcionar buenos resultados.

Un ejemplo más visual de esto se muestra en la Figura 5, en una campaña publicitaria que se sirve de dos spots distintos, uno de 10 segundos de duración y otro de 30. Ambos spots comienzan con el mismo frame, de tal forma que, sin el frame de *offset*, el programa no diferenciaría cual es el spot corto y largo.

Figura 5. Diferencia en los frames de *offset* entre dos spots publicitarios con el primer frame idéntico.

La base de datos estará compuesta de los vídeos con los anuncios, en formato ".avi" o ".mp4", un fichero de texto con las huellas del primer frame de cada anuncio, otro fichero de texto con las huellas del frame de offset de cada anuncio y, de manera opcional, el frame utilizado para calcular la huella de cada anuncio y una imagen representativa de la huella en blanco y negro.

Figura 6. Estructura de la base de datos de anuncios.

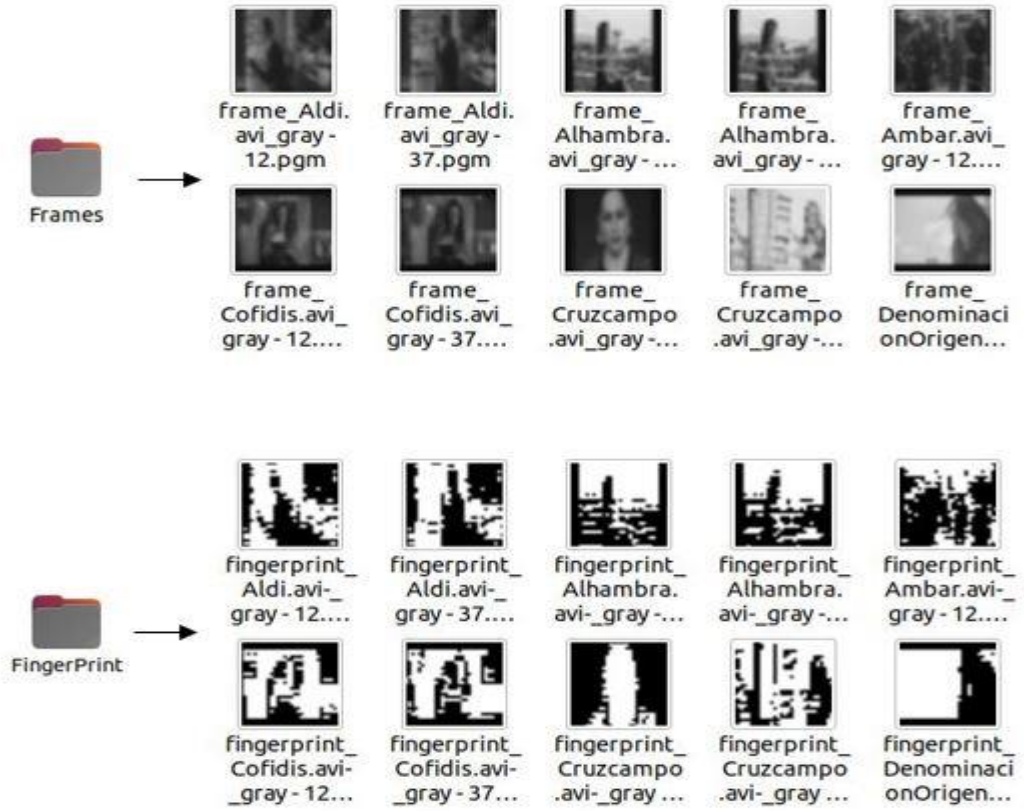

Figura 7. Contenido de las carpetas de la base de datos que guardan los frames utilizados y las imágenes representativas de las huellas.

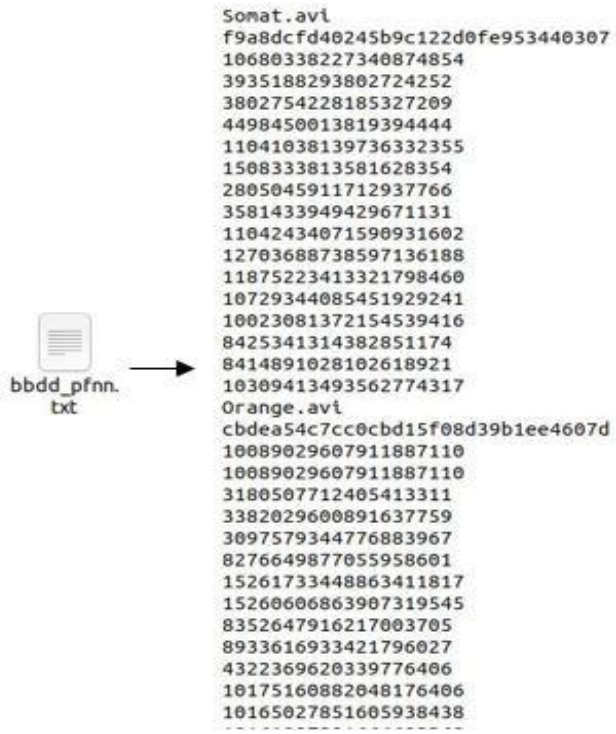

Figura 8. Contenido del fichero de texto que contiene las huellas de los anuncios.

2.2.3 Comparación entre huellas

En la etapa online, el programa principal comparará la huella del anuncio que está procesando en cada momento con la de la base de datos, evaluando la distancia de Hamming entre las huellas del primer frame de cada anuncio.

2.2.3.1 Distancia de Hamming

La distancia de Hamming es un algoritmo que, aplicado a dos cadenas de la misma longitud, devuelve el número de posiciones en las que ambas cadenas son diferentes entre sí. Dos cadenas serán idénticas si tienen una distancia de Hamming 0, y serán totalmente distintas si tienen una distancia de Hamming igual a su longitud. En la Figura 9 se muestran dos cadenas de 32 bits con una distancia de Hamming entre ellas igual a 5.

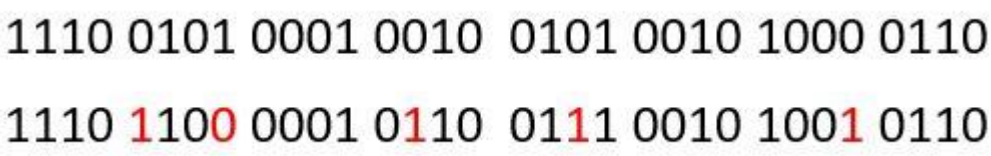

Figura 9. Distancia de Hamming d=5 entre dos cadenas de 32 bits.

El tamaño de las huellas que procesa el programa es de 1024 bits, y cuanto más grande sea el tamaño del hash, más susceptible a errores va a ser. Es necesario predeterminar un porcentaje mínimo de distancia Hamming para que dos hashes sean considerados iguales. A este porcentaje se le ha llamado umbral Hamming, y en función del método de hashing que se utilice, se debe configurar de una manera u otra (ver Capítulo 7).

Por ejemplo, si se fija el umbral Hamming al 10%, dos hashes de 1024 bits serán reconocidos como iguales si no tienen más de 102 posiciones distintas. Y, si se fija al 1%, no podrán diferir en más de 10 posiciones.

La aplicación SIRA-TV compara uno a uno el hash del primer frame no negro del anuncio en detección con los de la base de datos. Tras hacer esto, puede haber varios candidatos que estén por debajo del umbral Hamming, uno o ninguno. En el caso que solo haya un candidato se elige ese vídeo y se finaliza el proceso de búsqueda. Cuando hay varios candidatos o ninguno, se hace una nueva comparación con el frame de *offset* y, si lo hay, el candidato elegido será el que menor distancia Hamming presente en este caso.

Una vez se ha realizado la comparación los resultados pueden ser:

- El anuncio no es reconocido y no está en la base de datos
- El anuncio no es reconocido y sí está en la base de datos
- El anuncio es reconocido y corresponde con el original
- El anuncio es reconocido y no se corresponde con el original

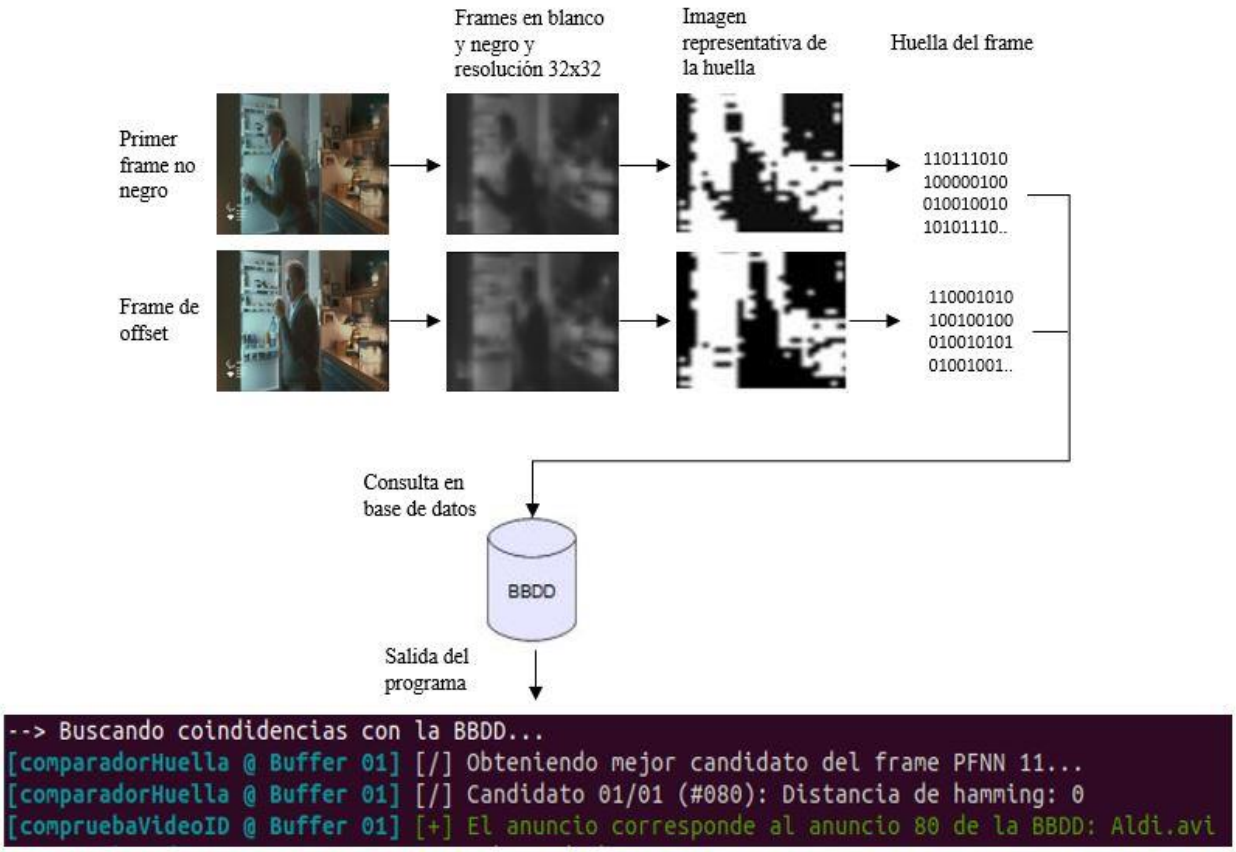

Figura 10. Ejemplo de extracción de huella y búsqueda en la base de datos.

En los casos que el anuncio no es reconocido, la aplicación SIRA-TV guardará dicho segmento de vídeo en memoria, para la posterior visualización y clasificación por parte del usuario. Esto es muy útil para actualizar la base de datos con nuevos anuncios.

Se buscará, el mejor algoritmo de hashing que garantice el máximo de resultados positivos (detecciones de anuncios verdaderas) y minimice los negativos (no haga detecciones o haga detecciones falsas). Para dicha elección, se tendrán en cuenta diferentes factores que puede sufrir la señal de televisión: ruido, disminución en la velocidad de fotogramas y otras distorsiones.

2.2.4 Cambios en la programación

Una vez explicada la naturaleza de las huellas y la base de datos de anuncios, queda explicar cómo será la detección de un anuncio en el streaming. Para esto, el programa detectará cualquier cambio en la programación cuando se sucedan uno o varios frames negros simultáneamente y comenzará el proceso de detección con el primer frame no negro después de esta secuencia de frames negros.

Un inconveniente es que SIRA-TV también va a procesar segmentos de programación que no correspondan con programación destinada a anuncios, esto puede suceder cuando se dé cualquier cambio entre tomas en una emisión o cuando la señal se vaya a negro por cualquier circustancia. La aplicación SIRA-TV se encargará de guardar estos segmentos en una carpeta específica distinta a la de los anuncios no reconocidos.

3 ENTORNO DE TRABAJO Y LIBRERÍAS UTILIZADAS

En algún lugar,
algo increíble está esperando a ser descubierto.
- Carl Sagan -

Para la realización del proyecto se han necesitado todos los programas informáticos que se detallan a continuación. La premisa en todo momento ha sido la de utilizar el software actualizado a la última versión. La aplicación SIRA-TV está programada en lenguaje C, se ha hecho uso del sistema operativo Ubuntu y del entorno de desarrollo Eclipse.

3.1 Oracle VM VirtualBox 6.1.22

Es un software de virtualización multiplataforma de código abierto. Permite ejecutar la máquina virtual Ubuntu y configurar parámetros como la memoria RAM, memoria de vídeo, controlador gráfico, etc., en función de las necesidades del usuario.

3.2 Máquina virtual Ubuntu 20.04.2 LTS

Hasta la fecha, la versión más actual de Ubuntu. El sistema operativo es de 64 bits al que se le ha dotado de 12GB de memoria RAM y 100GB de memoria de almacenamiento.

Sobre la máquina virtual se ha instalado el siguiente software:

3.2.1 Eclipse 4.18

Eclipse es un entorno de desarrollo software multi-lenguaje (en este caso se ha programado utilizando lenguaje C) que proporciona las herramientas necesarias para crear, compilar y depurar la aplicación. Permite trabajar con librerías de código abierto como las de OpenSSL o FFmpeg.

3.2.2 OpenSSL 1.1.1.f

OpenSSL es una librería de propósitos generales relacionados con la criptografía. Conoce el algoritmo ‘MD5’, un algoritmo criptográfico qué ante una entrada de tamaño arbitrario proporciona a la salida un hash de 128 bits.

3.2.3 FFMPEG 4.4

FFmpeg es un software libre basado en un conjunto de librerías que permiten realizar infinidad de tareas sobre archivos multimedia como multiplexar, demultiplexar, filtrar, codificar, decodificar, reproducir, editar, y un largo etcétera. Permite trabajar directamente desde la línea de comandos o a través de una aplicación correctamente programada. Además, es compatible con todo tipo de formatos multimedia.

Las librerías contenidas en su última versión son las siguientes:

- *libavutil*: Es la primera librería que se debe incluir en el proyecto ya que engloba funciones que utilizarán el resto de librerías de FFmpeg. Esta librería define la estructura *AVFrame*, que sirve para referenciar a los frames de un archivo multimedia. Se definen numerosas funciones matemáticas que se utilizarán para trabajar con las bases de tiempo de los archivos. Por último, *libavutil* contiene también la mayoría de códigos de error.

- *libavcodec*: Es la librería en la que más actualizaciones se han encontrado en comparativa con las versiones anteriores. Contiene las funciones necesarias para las tareas de codificación y decodificación, así como los distintos codecs de audio, vídeo y subtítulos que se utilizan. Define la estructura *AVPacket*, que sirve para guardar datos multimedia comprimidos, y el formato de píxel, ambos utilizados en el programa.

  Entre los módulos que la forman, destacan:

  Decoding: Incluye funciones para las tareas de decodificación como son *int avcodec_send_packet()*, que envía los paquetes al decodificador, e *int avcodec_receive_frame(),* que recibe los datos ya decodificados.

  Encoding: Análogamente al módulo anterior contiene las funciones utilizadas para codificación, con *int avcodec_send_frame()* se envían los frames al codificador y con *int avcodec_receive_packet()* se reciben los datos codificados.

- *libavformat:* Esta librería proporciona los multiplexores y demultiplexores para los formatos contenedores multimedia.

  Demultiplexar: Los demultiplexores leen un archivo multimedia y lo dividen en paquetes de datos, estos paquetes contienen uno o varios frames decodificados. Entre las funciones de este módulo se encuentran *avformat_open_input()*, *av_read_frame()*, *avformat_close_input()*.

  Multiplexar: Los multiplexores toman los datos codificados y los escriben en el formato de salida especificado.

- *libavdevice:* Proporciona a la librería anterior los dispositivos necesarios para sus funciones.

- *libavfilter:* Es una librería que contiene un amplio número de filtros para realizar operaciones sobre los frames.

- *libswresample:* Esta librería sirve para manipular archivos de audio pudiéndole realizar operaciones de muestreo o conversiones, entre otras. Contiene la función *int swr_convert()*, que convierte un archivo de audio al formato que se le especifique.

- *libswscale:* Librería que contiene funciones para conversión de colores en las imágenes.

```
miguel@miguel-VirtualBox:~/git/ffmpeg$ ./ffmpeg
ffmpeg version N-101481-gb87781649e Copyright (c) 2000-2021 the FFmpeg developers
  built with gcc 9 (Ubuntu 9.3.0-17ubuntu1~20.04)
  configuration: --enable-zlib
  libavutil      56. 68.100 / 56. 68.100
  libavcodec     58.130.100 / 58.130.100
  libavformat    58. 72.100 / 58. 72.100
  libavdevice    58. 12.100 / 58. 12.100
  libavfilter     7.109.100 /  7.109.100
  libswscale      5.  8.100 /  5.  8.100
  libswresample   3.  8.100 /  3.  8.100
Hyper fast Audio and Video encoder
usage: ffmpeg [options] [[infile options] -i infile]... {[outfile options] outfile}...
```

Figura 11. Vista en consola de las versiones de las librerías de FFmpeg.

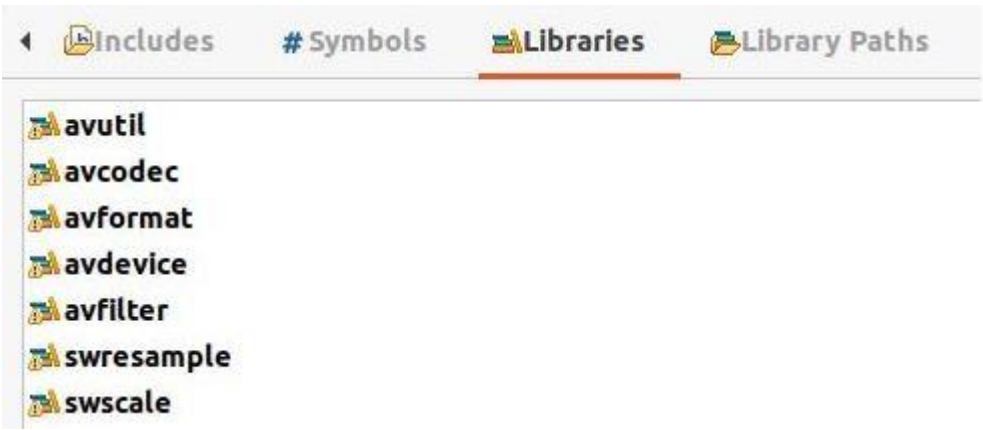

Figura 12. Librerías de FFmpeg incluidas en el proyecto de la aplicación SIRA-TV.

4 APLICACIÓN SIRA-TV

Sistema Inteligente de Reconocimiento de Anuncios de Televisión, en adelante SIRA-TV, es un software desarrollado por el Dpto. de Teoría de la Señal y Comunicaciones de la Universidad de Sevilla. El objetivo principal del programa es la detección de anuncios de televisión a tiempo real. Para identificar correctamente los anuncios calcula una huella de estos y la compara con las huellas contenidas en una base de datos (también creada por la misma aplicación). Si no encuentra ningún candidato, guarda el vídeo en memoria para que el usuario pueda visualizarlo y poder actualizar la base de datos.

La aplicación está formada por dos módulos distintos, uno encargado de la ingesta del streaming y la detección de los anuncios (etapa online), y otro encargado de la creación de la base de datos de anuncios (etapa offline).

A continuación, se describe brevemente la estructura y funcionamiento de la aplicación según los ficheros y funciones que forman los dos módulos del programa.

4.1 SIRA-TV-2.1

Última versión del programa antes del comienzo del presente proyecto. Su estructura, funciones y desarrollo están perfectamente detallados en el Trabajo de Fin de Grado de José Javier Alcántara Armenteros [13]. Sobre esta versión, de 2016, se han realizado todas las actualizaciones y mejoras explicadas en los siguientes apartados. Este módulo se encarga de la ingesta de streaming, detección de anuncios, comparación de anuncios con la base de datos y guardar los fragmentos de vídeo no reconocidos.

4.1.1 sira-tv-2.1.c

Fichero que contiene la función *main* del programa, desde esta se invoca a la función *void initDeteccion()*, que se encarga de inicializar el proceso de detección de anuncios de un streaming.

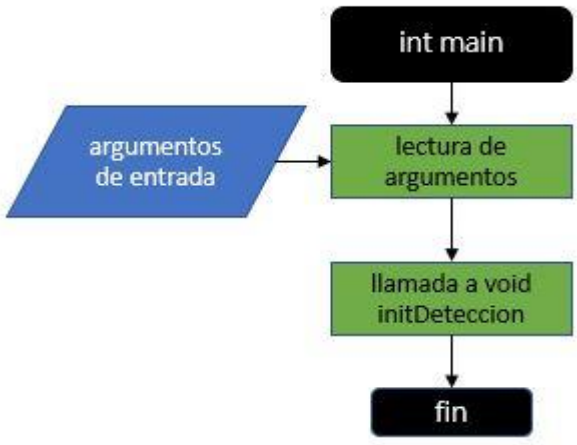

Figura 13. Diagrama de flujo de sira-tv-2.1.c

4.1.2 sira-tv-thread.c

Este fichero contiene el hilo principal del programa. Al principio, en la función *initDeteccion()* se hace una llamada a la función *void procesaStream(),* que primero inicializa las variables y contextos del programa empleando la función *void initSiraContexts()*, y a continuación carga la base de datos en memoria con *void* initBBDD. Por último, ejecuta la rutina principal del programa, que funciona de la siguiente manera:

> La rutina principal recibe un paquete de datos procedente del streaming, lo clasifica según sea de vídeo o de audio y lo decodifica, si el frame decodificado es de vídeo calcula si es un frame negro. En caso positivo, activa una bandera que indica que el último frame leído era negro. Al decodificar los siguientes paquetes de vídeo, cuando se encuentre ante el primer frame no negro, se procesa dicho frame como el primer frame de un anuncio, y procede a guardar todos los frames posteriores en el *buffer* hasta que esté ante otro frame negro o hasta que se supere la duración del anuncio preestablecida.
>
> Al mismo tiempo que guarda los frames en el *buffer*, llama a la función *void initBusqueda(),* que con el primer frame no negro y con el frame de *offset*, calcula la huella y las compara con la de la base de datos de anuncios utilizando la función *void comparadorHuella().* Por último, la función *void compruebaVideoID*, gestiona el resultado de la comparación de las huellas, clasificando el anuncio en *encontrado* o *no encontrado* y si no lo ha encontrado, manda la información del *buffer* al multiplexor para codificar el vídeo y guardarlo en el disco con la función *void saveVideo()*.

Una vez concluye la búsqueda, se liberan los recursos utilizados con *void destroyBusqueda()*.

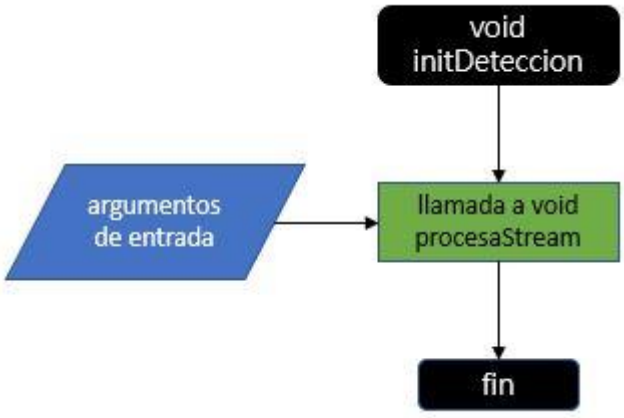

Figura 14.1. Diagrama de flujo de sira-tv-thread.c (función *initDeteccion*).

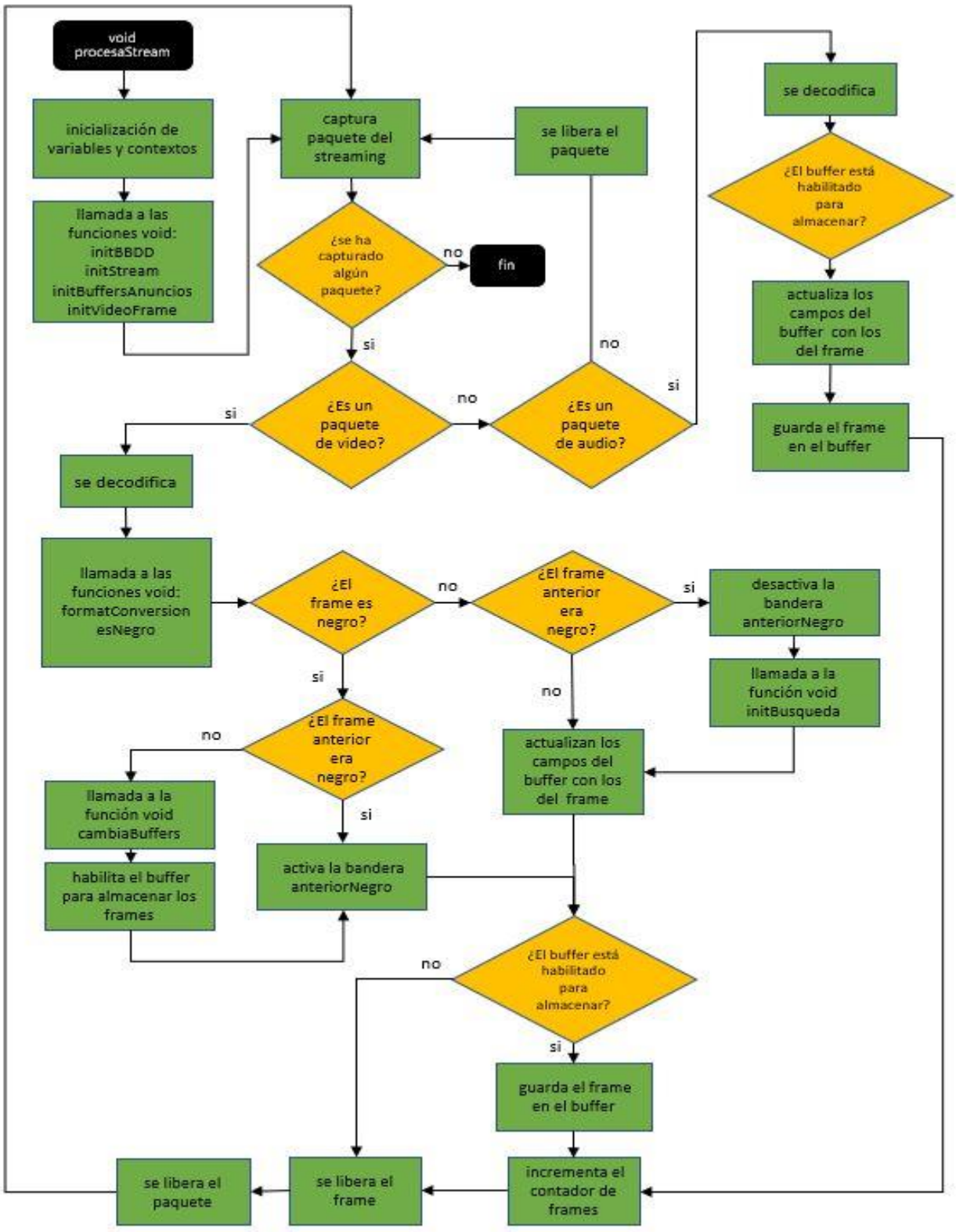

Figura 14.2. Diagrama de flujo de sira-tv-thread.c (función *procesaStream*).

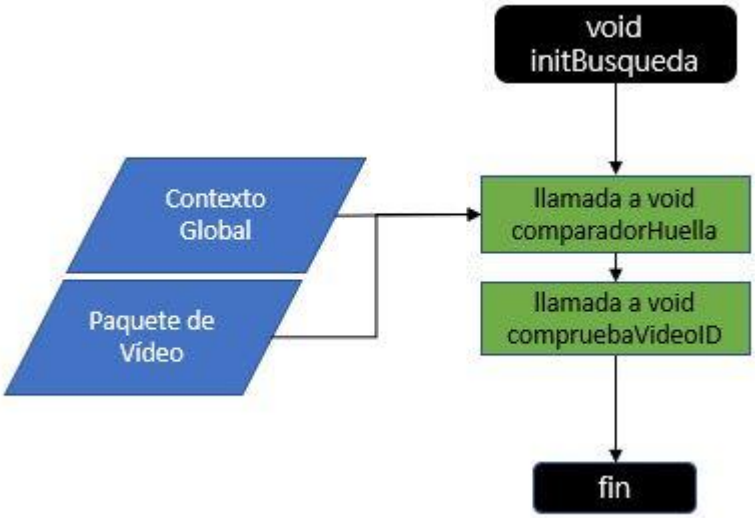

Figura 14.3. Diagrama de flujo de sira-tv-thread.c (función *initBusqueda*).

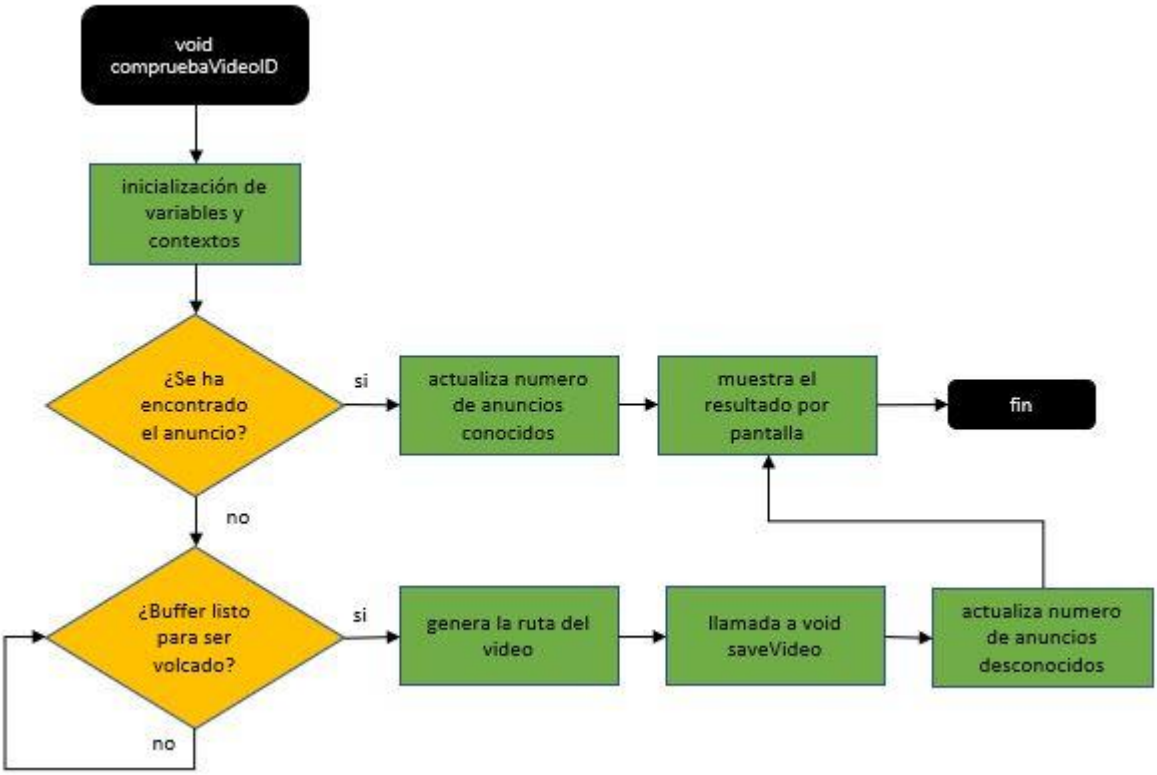

Figura 14.5. Diagrama de flujo de sira-tv-thread.c (función *compruebaVideoID*).

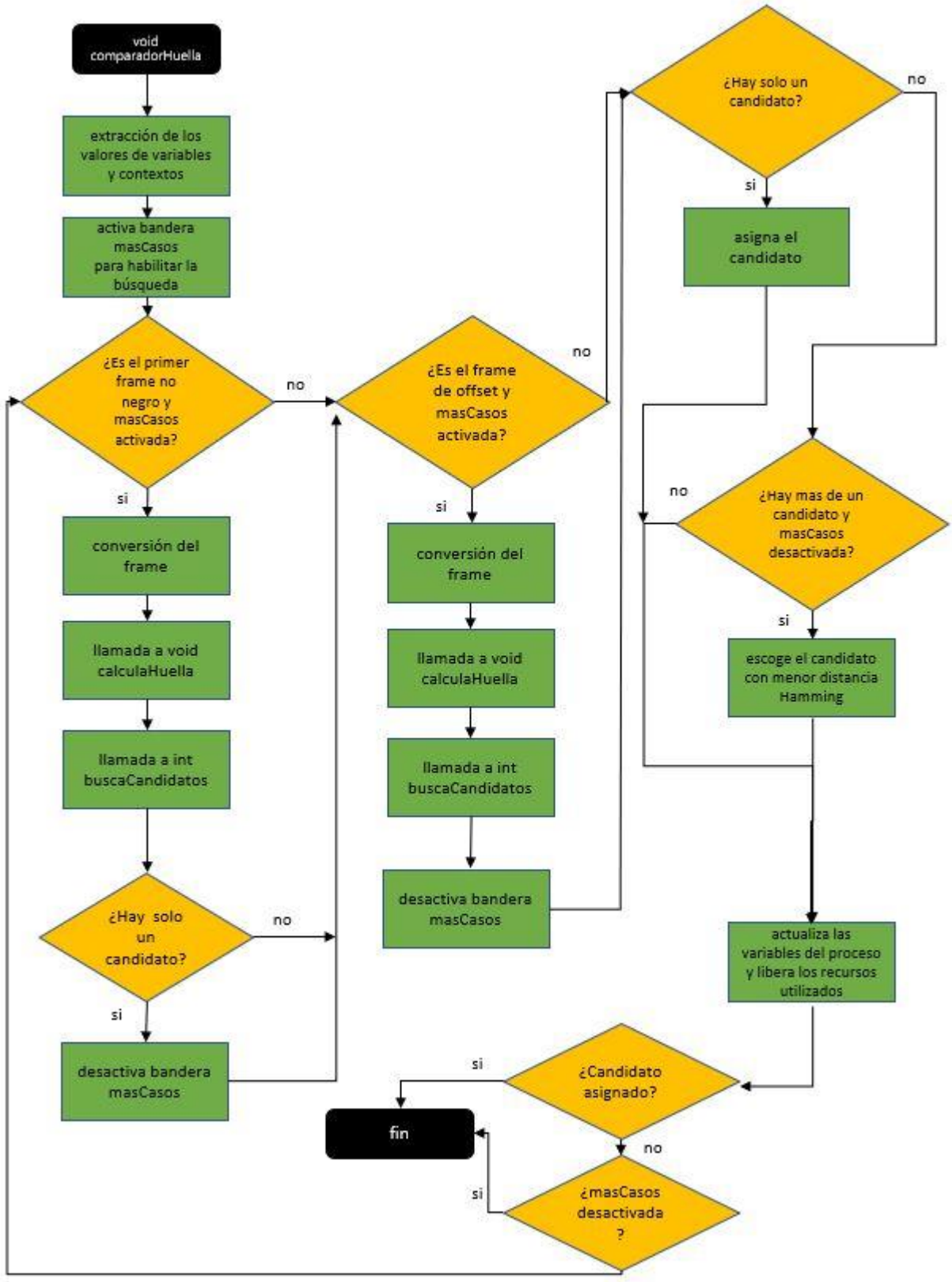

Figura 14.4. Diagrama de flujo de sira-tv-thread.c (función *comparadorHuella*).

4.1.3 sira-tv-muxer.c

Desde la función *void comparadorHuella()* del fichero anterior se invoca a la función *void saveVideo()* de este fichero. Esta se encarga de guardar el fragmento de vídeo cuyo *hash* o huella no corresponde con ningún candidato de la base de datos.

En primer lugar inicializa los códecs para la salida usando *void addStream()* y los abre utilizando *void open_video()* y *void open_audio()*. Una vez tiene el codificador preparado, procesa la información del *buffer* que contiene todos los frames de vídeo y audio del anuncio no reconocido, utilizando *write_audio_frame()* y *write_video_frame()* para codificar y escribir en memoria el frame de audio o vídeo respectivamente.

Finalmente, se libera el *buffer* y los recursos utilizando *void close_stream()*.

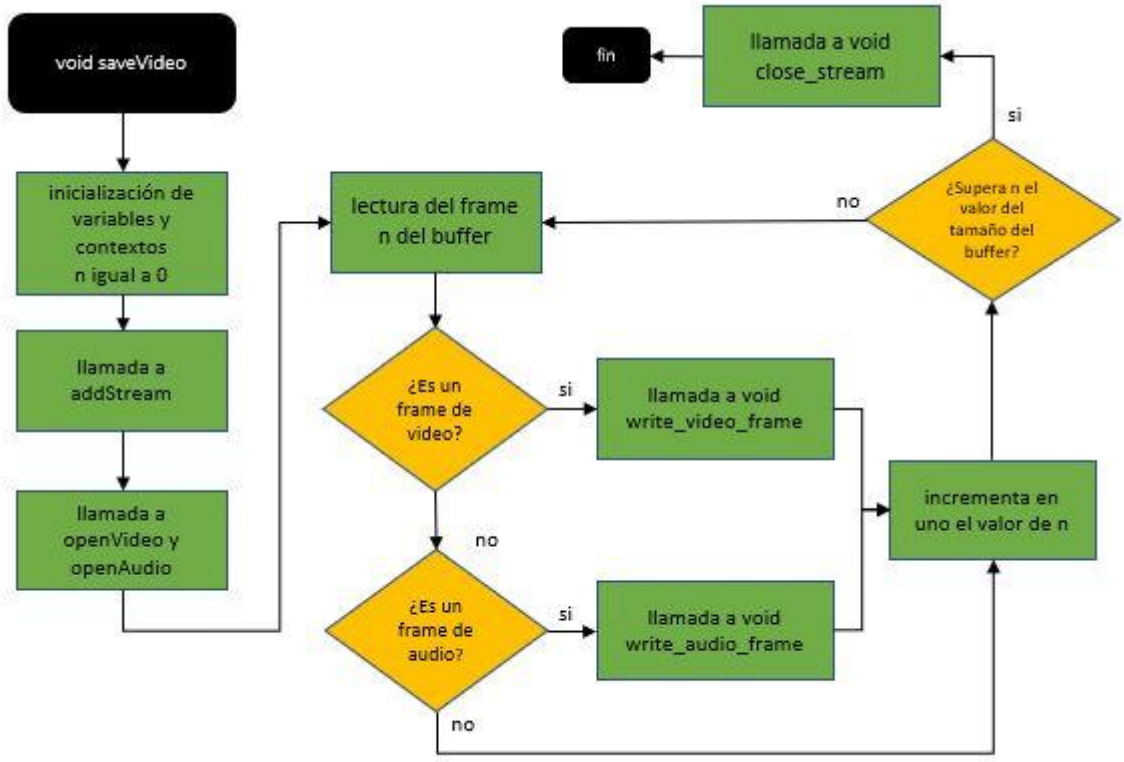

Figura 15. Diagrama de flujo de sira-tv-muxer.c

4.1.4 sira-tv-huellas.c

Desde la función *void comparadorHuella()* de *sira-tv-thread* se hace uso de la función de este fichero, *void calculaHuella(),* para calcular la huella del primer frame no negro y del frame de *offset* del anuncio que la rutina principal está detectando en ese instante.

Una vez se ha calculado, la rutina principal también hace uso de la función *int buscaCandidatos()*, que devuelve el número de candidatos encontrados al consultar la base de datos.

Los candidatos son seleccionados por su distancia de Hamming con respecto a la huella original. Se define un umbral mínimo de error en esta distancia para determinar si un vídeo es candidato o no, con lo que puede haber varios candidatos, uno o ninguno. En el caso que haya varios, el programa elige el que menor distancia de Hamming presente.

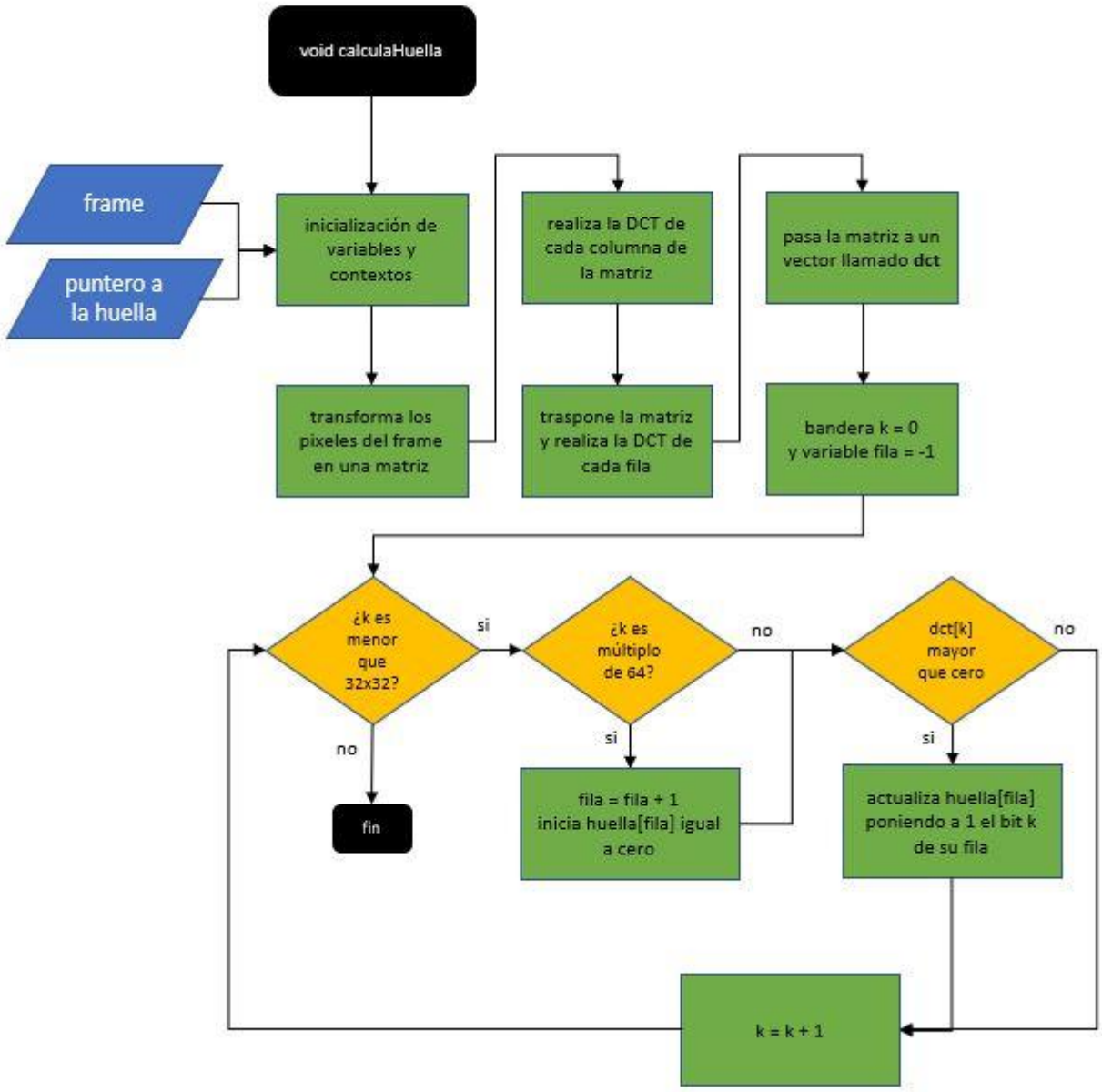

Figura 16.1. Diagrama de flujo de sira-tv-huellas.c (función *calculaHuella*).

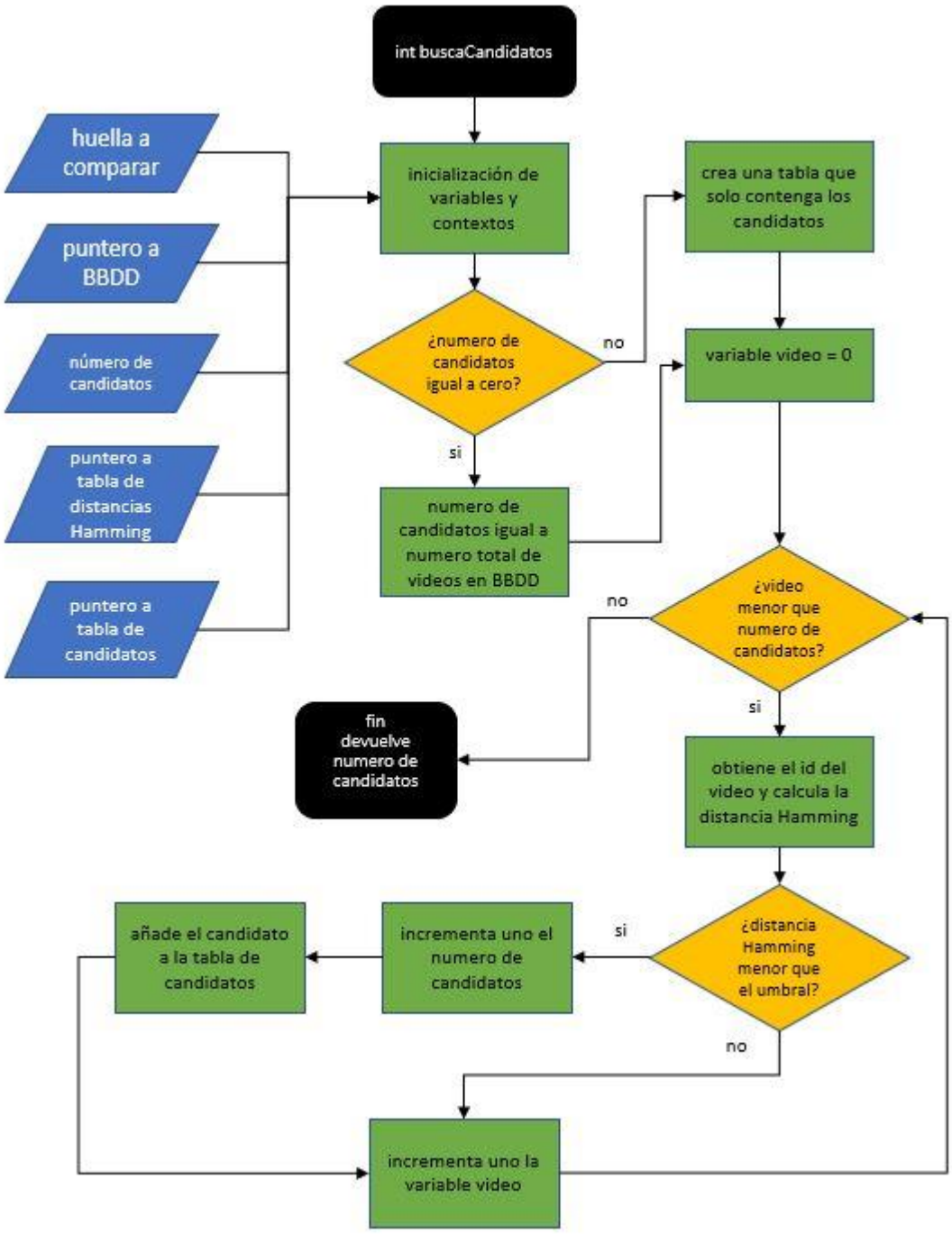

Figura 16.2. Diagrama de flujo de sira-tv-huellas.c (función *buscaCandidatos*).

4.1.5 sira-tv-aux.c

En este fichero están contenidas las funciones más genéricas del programa, que se citan a continuación:

- *void loadArgs()*: función que gestiona los argumentos de entrada.
- *void initLog()*: función que inicializa la salida de información, puede ser en un fichero o en consola.
- *int formatoValido()*: detecta si el fichero de entrada tiene un formato válido.

- *int esNegro()*: recibe un frame y calcula si es negro o no en función de su porcentaje de píxeles negros.
- *void saveFrame()*: esta función guarda en disco un frame en formato *.ppm* o *.pgm*.
- *void initStream()*: intenta abrir un archivo de entrada, recuperando la información de los streams que alberga, localizando el stream de vídeo y abriendo el códec necesario para decodificarlo.
- *void initVideoFrame()*: fija el formato de un frame y ubica el *buffer* necesario para trabajar con él.
- *void initAudioFrame()*: fija el formato, canal y número de muestras de un frame de audio y ubica el *buffer* necesario para trabajar con él.
- *void formatConversion()*: función que convierte un frame origen a otro de salida basándose en su formato y dimensión.
- *long long timeval_diff()*: calcula la diferencia entre dos instantes de tiempo en microsegundos.
- *void initBuffersAnuncios()*: reserva la memoria suficiente para albergar el anuncio que está siendo analizado.
- *void freeBufferAnuncios()*: libera la memoria reservada en la función anterior.
- *void freeSiraGlobalContext()*: función que libera la memoria reservada por los contextos del programa.
- *void cambiaBuffers()*: cambia los identificadores de los *buffers* de entrada y salida.
- *void infoHelp()*: muestra a la salida toda la información del programa en relación con los argumentos de entrada y su uso.

4.1.6 sira-tv-bbdd.c

Es un fichero con funciones para la gestión de la base de datos de anuncios. Desde *sira-tv_thread* se invoca a la función *initBBDD()* para inicializar la base de datos de huellas a partir de los directorios de entrada.

Esta función llama a las funciones *void cuentaVideos()* y *void loadBBDD()* para contar el número de vídeos que conforman la base de datos y cargar la base de datos en memoria respectivamente.

4.1.7 sira-tv.h

Es el fichero que contiene las librerías y definiciones de tipos, datos, funciones y estructuras que se utilizan en todo el programa.

Tabla 4-1. Definición de tipos de datos y estructuras de SIRA-TV

typedef	Nombre	Uso
unsigned long long int	huella_t	Tipo usado para almacenar parte de la huella (64 bits)
struct (Tabla 4-2)	bbdd_t	Base de datos
struct (Tabla 4-3)	bufferAnuncios	*Buffer* que almacena frames de audio y vídeo
struct (Tabla 4-4)	OutputStream	Para escribir el flujo de bytes en el stream de salida
struct	controlStream	Contiene las variables de control del programa
struct	siraDecoContext	Contexto del decodificador del streaming
struct	siraInfoContext	Contexto de la información de entrada al programa
struct	siraBBDDContext	Contexto de la base de datos
struct	siraProcContext	Contexto del procesado del streaming
struct	siraThreadsContext	Contexto de los hilos del programa
struct (Tabla 4-5)	siraGlobalContext	Contexto global del programa

Tabla 4-2. Campos de la estructura *bbdd_t*

Tipo	Nombre	Uso
char[SIRAPATH_MAX]	name	Nombre de la base de datos
huella_t **	huellas	Tabla de huellas de los vídeos
int	offset	Número del frame a partir del primer frame no negro

Tabla 4-3. Campos de la estructura *bufferAnuncios*

Tipo	Nombre	Uso
volatile int	state	Estado del *buffer*
char *	output	Nombre del fichero de salida
int	framesContenidos	Número de frames contenidos por el *buffer*
long log int	pfnn, osfnn, ufnn	Índice del primer frame no negro, frame de offset y último frame no negro
long long unsigned int	rpfnn, rsofnn, rufnn	Frame relativo al primer frame no negro, frame de offset y último frame no negro
long long int	ufa	Índice del ultimo frame de audio del *buffer*
long long int	vfpts, afpts, vfdts, afdts	Valor de 'pts' y 'dts' del primer frame de vídeo y audio almacenado
frameAnuncio*	VideoBuffer, AudioBuffer	*Buffer* del segmento de vídeo o audio siendo procesado
AVFrame **	pFramesOrdenados	Tabla de direcciones a los frames en orden de llegada

Tabla 4-4. Campos de la estructura *OutputStream*

Tipo	Nombre	Uso
AVStream *	st	Estructura con información del stream
int64_t	next_pts	Campo 'pts' del siguiente frame que se generará
int	samples_count	Número de muestras de audio
AVFrame*	frame, tmp_frame	Punteros al frame y al frame auxiliar
float	t, tincr, tincr2	Tiempos para sincronización
AVCodecContext *	enc	Contexto del codificador de salida
struct SwsContext *	sws_ctx	Estructura para utilizar la librería swscale
struct SwrContext *	swr_ctx	Estructura para utilizar la librería swresample

Tabla 4-5. Campos de la estructura *siraGlobalContext*

Tipo	Nombre	Uso
siraBBDDContext*	bbddContext	Variables relacionadas con la base de datos
siraDecoContext*	decoContext	Variables relacionadas con el decodificador del streaming
siraInfoContext*	infoContext	Variables relacionadas con la información de entrada/salida
siraThreadsContext*	threadsContext	Variables relacionadas con los hilos
siraProcContext*	procContext	Variables relacionadas con el procesado del streaming
bufferAnuncios*	t_buffers	Tabla de *buffers*
controlStream*	ctrlStream	Control del streaming

4.2 SIRA-TV-FINGERPRINTER-0.4

SIRA-TV-FINGERPRINTER-0.4: Módulo encargado de crear una base de datos que contenga las huellas de los anuncios que la conforman. Para el cálculo de la huella, utiliza un método de hashing perceptual de imágenes aplicando la transformada de coseno discreta (DCT).

4.2.1 sira-tv-fingerprinter-0.4.c

Fichero que contiene la función int *main()* del programa, declara e inicializa los contextos y parámetros necesarios para su ejecución y carga los argumentos de entrada con *void loadArgs()*. Invoca a la función *void initBBDD()* para comenzar con el proceso de creación de la base de datos de anuncios.

4.2.2 sira-tv-bbdd.c

Contiene las funciones para la creación y gestión de la base de datos de anuncios. Desde la función *main* del fichero anterior se llama a la función *void initBBDD()*, que inicializa la base de datos de huellas a partir de los directorios de entrada haciendo uso de la función *void creaBBDD()* para crear un archivo de base de datos desde cero.

En *void creaBBDD()* se leen uno a uno todos los vídeos que hay en el directorio, se comprueba si el formato de los vídeos es válido, reserva memoria suficiente y se llama a la función *void extraeHuella()* encargada de calcular las huellas de los vídeos.

La función *void extraeHuella()* obtendrá la huella del vídeo que recibe como entrada, para esto, en primer lugar decodifica la entrada de vídeo, a continuación obtiene el primer frame no negro y el frame de *offset* (25 frames después), y por último hace uso de la función *void calculaHuella()* para obtener la huella de dichos frames.

4.2.3 sira-tv-huellas.c

Es un fichero con dos funciones auxiliares para la gestión de las huellas.

La función *void calculaHuella()* recibe un frame y calcula su huella de 1024 bits, y la función *void saveFingerprint()* guarda una imagen de la huella en formato *'ppm'*.

4.2.4 sira-tv-hash.c

Este fichero sirve para otorgarle a las huellas de cada anuncio un *hash* MD5. Utilizando las librerías de OpenSSL, la función *void getHash()* recibe a su entrada las huellas de 1024 bits de todos los vídeos de la base de datos y las codifica en unas huellas de 128 bits que se añaden de forma adicional a la base de datos.

4.2.5 sira-tv-aux.c

Este fichero es idéntico al explicado en el apartado 4.4.1.5 *sira-tv_aux.c* ya que las funciones son genéricas para los dos programas.

4.2.6 sira-tv.h

Es el fichero que contiene las librerías y definiciones de tipos, datos, funciones y estructuras que se utilizan en todo el programa, es igual al fichero descrito en el apartado 4.1.1.7 *sira-tv.h*.

5 ACTUALIZACIÓN DE LA APLICACIÓN SIRA-TV

Sólo en la oscuridad puedes ver bien las estrellas.

- Martin Luther King -

Este capítulo está dedicado a explicar cómo se ha llevado a cabo el proceso de actualización del programa y las novedades que se han introducido. Tal y como se indica en el apartado anterior, la última versión era del 2016, lo que significa que un amplio número de funciones de FFmpeg que utiliza se han quedado obsoletas. La versión 2.5 de FFmpeg que emplea dista mucho con la versión 4.4 (la más moderna hasta la fecha). Una vez se comprobó el correcto funcionamiento del programa actualizado se introdujeron algunas novedades en el programa.

5.1 Actualización de funciones y librerías de FFmpeg

Para actualizar el código fuente ha sido de vital importancia el manual y la documentación acerca de las funciones que FFmpeg tiene a disposición en su web oficial. [14]

Cuando FFmpeg actualiza una función o un tipo de dato de una versión a otra, el elemento no es borrado de sus librerías, si no que se le asigna un tipo especial llamado *attribute_deprecated* (Figura 17). A la hora de compilar el programa, todas las funciones obsoletas que se estén utilizando aparecerán en la consola en forma de *warning* gracias a este tipo de dato especial. Es importante tener esto en cuenta ya que se puede creer que un programa con funciones de FFmpeg está funcionando correctamente cuando en realidad hay partes de él que no están siendo ejecutadas con normalidad.

```
../sira-tv-fingerprinter-0.4.c: In function ‘main’:
../sira-tv-fingerprinter-0.4.c:97:2: warning: ‘av_register_all’ is deprecated [-Wdeprecated-declarations]
   97 |  av_register_all();     // Registramos todos los codecs
      |  ^~~~~~~~~~~~~~~
In file included from /home/miguel/git/ffmpeg/libavdevice/avdevice.h:51,
                 from ../sira-tv.h:21,
                 from ../sira-tv-fingerprinter-0.4.c:64:
/home/miguel/git/ffmpeg/libavformat/avformat.h:1950:6: note: declared here
 1950 | void av_register_all(void);
```

Figura 17. Resultado de compilar el proyecto con una función de FFmpeg obsoleta.

5.1.1 SIRA-TV-2.1

Sobre el módulo que contiene el programa principal para la detección de los anuncios se han tenido que realizar actualizaciones a los ficheros que se indican a continuación.

5.1.1.1 sira-tv-2.1.c

Actualizaciones:

Se ha eliminado la línea de código que llama a la función *av_register_all()* para registrar los códecs porque está obsoleta, no ha necesitado ser remplazada.

Tabla 5–1-1. Funciones y tipos obsoletos de FFmpeg y sus alternativas de código en sira-tv-2.1.c

FFmpeg 2.5	FFmpeg 4.4
av_register_all()	no es necesario remplazar

5.1.1.2 sira-tv-thread.c

Actualizaciones:

*void *procesaStream()*: la función *avcodec_decode_video2()* utilizada anteriormente para decodificar un paquete está obsoleta, para realizar la decodificación correcta de un paquete de vídeo se ha sustituido dicha función por las funcion *avcodec_send_packet()*, que envía un paquete de datos al decodificador, y la función *avcodec_receive_frame()*, que devuelve los datos decodificados. Los fragmentos de código (Código 5-1-1-a y 5-1-1-b) muestran respectivamente las líneas de código del programa antes y después de las modificaciones.

```
. . .
if (packet.stream_index == video_stream_index) {
          avcodec_decode_video2(sDC->video_dec_ctx, pFrame, &got_frame,
&packet);
         if (got_frame) {
         sPC -> framesProcesados++;
           . . .
```

Código 5-1-1-a. Decodificación con *avcodec_decode_video2() [deprecated]*.

```
. . .
if (packet.stream_index == video_stream_index) {
          ret = avcodec_send_packet(sDC->video_dec_ctx, &packet);
         if (ret < 0) {
                   fprintf(stderr, "Error sending a packet to the decoder:
%s\n", av_err2str(ret));
                   siraThreadExit(sGC,1);
         }
        while (ret >= 0) {
                  ret = avcodec_receive_frame (sDC -> video_dec_ctx, pFrame);
                  if (ret == AVERROR(EAGAIN) || ret = AVERROR_EOF)
                            break;
                 else if (ret < 0){
                                fprintf(stderr, "Error receiving a frame:
%s\n", av_err2str(ret));
                               siraThreadExit(sGC,1);
                   }
         sPC -> framesProcesados++;
           . . .
```

Código 5-1-1-b. Decodificación con *avcodec_send_packet()* y *avcodec_receive_frame()*

De la misma manera, para la decodificación del paquete de audio se utilizaba la función *avcodec_decode_audio4()*, que se ha tenido que sustituir por *avcodec_send_packet()* y *avcodec_receive_frame()*.

Cuando el programa termina de procesar el paquete, lo libera. En la versión anterior utilizaba la función *av_free_packet()*, que se ha tenido que actualizar por *av_packet_unref()*.

Tabla 5-1-2. Funciones y tipos obsoletos de FFmpeg y sus alternativas de código en sira-tv-thread.c

FFmpeg 2.5	FFmpeg 4.4
avcodec_decode_video2()	avcodec_send_packet() / avcodec_receive_frame()
avcodec_decode_audio4()	avcodec_send_packet() / avcodec_receive_frame()
av_free_packet()	av_packet_unref()

5.1.1.3 sira-tv-muxer.c

Actualizaciones:

- Se ha eliminado la función *void initMuxer()* y se ha volcado su contenido en la misma función *void saveVideo()*.
- El tipo de estructura *codec* está obsoleto en la nueva versión de FFmpeg, con lo que ya no se puede utilizar el campo *codec* de las estructuras *OutputStreaming*, se hará uso de la estructura *AVCodecContext.*
- Las banderas *CODEC_CAP_VARIABLE_FRAME_SIZE* y *CODEC_FLAG_GLOBAL_HEADER*, han cambiado de nombre, ahora se llaman *AVCODEC_CAP_VARIABLE_FRAME_SIZE* y *AVCODEC_FLAG_GLOBAL_HEADER* respectivamente.
- *int write_video_frame()*: el campo *'pts'* de la estructura AVStream ya no existe, se han eliminado las líneas de código que lo utilizaban y se ha cambiado notablemente la estructura de la función.
- *int write_video_frame():* la función *avcodec_encode_video2()* que sirve para mandar los frames de vídeo al codificador está obsoleta. Se ha sustituido por las funciones *avcodec_send_frame()* para enviar el frame al codificador y *avcodec_receive_packet()* para recibir el paquete codificado. Los fragmentos de código (Código 5-1-2-a y 5-1-2-b) muestran respectivamente las líneas de código del programa antes y después de las modificaciones.
- *int write_audio_frame():* análogo al caso del frame de vídeo, el campo '*pts*' de la estructura AVStream ya no existe y se han eliminado las líneas de código que lo invocaban.
- *int write_audio_frame():* la función *avcodec_encode_audio2()* empleada para mandar los frames de audio al codificador está obsoleta. Se ha sustituido por las funciones *avcodec_send_frame()* para enviar el frame al codificador y *avcodec_receive_packet()* para recibir el paquete codificado.
- *avcodec_close()* ya no se puede utilizar para dejar cerrar el codificador porque como se ha indicado anteriormente, la estructura *codec* está obsoleta. Se ha reemplazado por la función *avcodec_free_context()* que libera la estructura *AVCodecContext*.

```
. . .
ret = avcodec_encode_video2(c, &pkt, ost->frame, &got_packet);
if (ret<0) {
          fprintf(stderr, "Error encoding video frame: %s\n",
av_err2str(ret));
          siraThreadExit(sGC,1);
if (got_packet) {
          ret = write_frame(oc, &c->time_base, ost->st, &pkt);
}
. . .
```

Código 5-1-2-a. Codificación con *avcodec_encode_video2() [deprecated]*.

```
. . .
ret = avcodec_send_frame(c, ost.>frame);
if (ret<0) {
          fprintf(stderr, "Error encoding video frame: %s\n",
av_err2str(ret));
          siraThreadExit(sGC,1);
}
ret = avcodec_receive_packet(c, &pkt);
if (!ret) {
ret = write_frame(oc, &c->time_base, ost->st, &pkt);
}
. . .
```

Código 5-1-2-b. Codificación con *avcodec_send_frame()* y *avcodec_receive_packet().*

Tabla 5–1-3. Funciones y tipos obsoletos de FFmpeg y sus alternativas de código en sira-tv-muxer.c

FFmpeg 2.5	FFmpeg 4.4
codec	AVCodecContext
CODEC_CAP_VARIABLE_FRAME_SIZE	AVCODEC_CAP_VARIABLE_FRAME_SIZE
CODEC_FLAG_GLOBAL_HEADER	AVCODEC_FLAG_GLOBAL_HEADER
AVStream campo 'pts'	sin remplazo
avcodec_encode_video2()	avcodec_send_frame() y avcodec_receive_packet()
avcodec_encode_audio2()	avcodec_send_frame() y avcodec_receive_packet()
avcodec_close()	avcodec_free_context()

5.1.1.4 sira-tv-aux.c

Actualizaciones:

- *void_initStream()*: esta función intenta abrir un archivo de entrada, recuperando información de los streams que alberga y abriendo el códec necesario para decodificarlo. Se ha eliminado la línea de código que obtenía un puntero al contexto de códec del stream de vídeo (Código 5-1-3.a).

```
. . .
   sGC->decoContext->video_dec_ctx = sGC->decoContext->video_stream->codec
 . . .
```

Código 5-1-3-a. Apertura del decodificador de vídeo utilizando la estructura *codec [deprecated]*.

Como se ha explicado anteriormente, la estructura *codec* no existe en la nueva versión de FFmpeg y hay que utilizar la estructura *codecpar*. Para hacer esto se han añadido las líneas de código:

```
. . .
      sGC->decoContext->vídeo_dec_ctx=avcodec_alloc_context3(dec);
      avcodec_parameters_to_context(sGC->decoContext->video_dec_ctx,
      (*fmt_ctx)->streams[*v_i]->codecpar);
. . .
```

Código 5-1-3-b. Apertura del decodificador de vídeo utilizando la estructura *codecpar*.

Análogamente, en el audio, se ha eliminado la línea que utilizaba la estructura *codec*:

```
. . .
   sGC->decoContext->audio_dec_ctx = sGC->decoContext->audio_stream->codec
 . . .
```

Código 5-1-3-c. Apertura del decodificador de audio utilizando la estructura *codec [deprecated]*.

Y se ha sustituido por las líneas:

```
. . .
      sGC->decoContext->audio_dec_ctx=avcodec_alloc_context3(dec);
      avcodec_parameters_to_context(sGC->decoContext->audio_dec_ctx,
      (*fmt_ctx)->streams[*a_i]->codecpar);
. . .
```

Código 5-1-3-d. Apertura del decodificador de vídeo utilizando la estructura *codecpar*.

- *void initVideoFrame():* inicializa y ubica un frame de vídeo según los parámetros de entrada (formato, altura y anchura). Para ubicar el frame en memoria utilizaba la función *avpicture_get_size()*, esta función está obsoleta y se ha cambiado por la función *av_image_get_buffer_size()*.

Tabla 5-1-4. Funciones y tipos obsoletos de FFmpeg y sus alternativas de código en sira-tv-aux.c

FFmpeg 2.5	FFmpeg 4.4
codec	avcodec_alloc_context3() y avcodec_parameters_to_context()
avpicture_get_size()	av_image_get_buffer_size()

5.1.1.5 sira-tv.h

Sobre el archivo de cabecera ha sido necesario modificar las siguientes estructuras:

- *struct OutputStream:* se le ha añadido el campo *AVCodecContext *enc* para poder utilizar el contexto del codificador en *sira-tv-muxer.c*.

Tabla 5-1-5. Funciones y tipos obsoletos de FFmpeg y sus alternativas de código en sira-tv.h

FFmpeg 2.5	FFmpeg 4.4
No se implementa	AVCodecContext

5.1.2 SIRA-TV-FINGERPRINTER-0.4

En este módulo, encargado de calcular la huella de los vídeos y crear la base de datos, se han realizado las actualizaciones de las librerías de FFmpeg a los ficheros que se indican a continuación.

5.1.2.1 sira-tv_bbdd.c

Actualizaciones:

void extraeHuella(): esta función se encarga de decodificar los paquetes de vídeo, leer sus frames y extraer la huella. Para ello, utilizaba la función *avcodec_decode_video2()* que está obsoleta y se ha remplazado por *avcodec_send_packet()* para envíar el paquete al a entrada del decodificador, y *avcodec_receive_frame()* para recibir el frame procedente del decodificador.

Por último, se ha sustituido la función *av_free_packet()* por *av_packet_unref()* para liberar el paquete una vez ha sido procesado.

Tabla 5–2-1. Funciones y tipos obsoletos de FFmpeg y sus alternativas de código en sira-tv-bbdd.c

FFmpeg 2.5	FFmpeg 4.4
avcodec_decode_video2() av_free_packet()	avcodec_send_packet() / avcodec_receive_frame() av_packet_unref()

5.2 Actualización de otras funciones del programa

Con el objetivo de intentar sacarle el máximo partido al programa, se han realizado otro tipo de actualizaciones para mejorar la calidad final de los vídeos.

5.2.1 Aumento de la resolución

Cuando el programa principal no es capaz de detectar un anuncio, manda la información que ha guardado en el *buffer* al multiplexor para guardar ese segmento de vídeo en memoria.

La versión anterior del programa enviaba los frames de vídeo al multiplexor con una resolución de 32x32 píxeles. Con esto conseguía ahorrar espacio en memoria, pero los resultados en la visualización del vídeo eran muy pobres.

En la nueva versión, se ha cambiado el tamaño del *buffer* reservado a los vídeos no reconocidos a su resolución inicial. Por lo general se ha trabajado con vídeos de una resolución de 720x576, con lo que se podrá visualizar el segmento de vídeo no reconocido mucho mejor. En la Figura 18 se muestra como el programa guardaba los vídeos en el disco antes y como los guardará a partir de la actualización.

Figura 18. Frame de un anuncio con resoluciones de 32x32 (izquierda) y 720x576 (derecha) píxeles.

Para realizar este cambio se ha tenido que modificar el fichero *sira-tv-thread.c* cambiando las líneas de código que guardan el frame en el *buffer*:

```
. . .
av_frame_copy(sGC->t_buffers[sPC->idBuffer].VideoBuffer[frameAnuncio].frame, pFrameYUV);

av_frame_copy_props(sGC->t_buffers[sPC->idBuffer].VideoBuffer[frameAnuncio].frame,
pFrameYUV);

. . .
```

Código 5-2-1.a. Envío del frame *pFrameYUV* (resolución: 32x32 píxeles) al *buffer*.

Por las siguientes líneas de código:

```
. . .
av_frame_copy(sGC->t_buffers[sPC->idBuffer].VideoBuffer[frameAnuncio].frame, pFrame);

av_frame_copy_props(sGC->t_buffers[sPC->idBuffer].VideoBuffer[frameAnuncio].frame,
pFrame);

. . .
```

Código 5-2-1-b. Envío del frame *pFrame* (resolución original) al *buffer*.

En dónde las funciones *av_frame_copy()* y *av_frame_copy_props()* copian los datos del frame origen (segundo parámetro de la función) al frame destino (primer parámetro). Los frames origen y destino tienen que haber sido previamente inicializados, con lo que en la función *void initBuffersAnuncios()* se ha cambiado la forma de inicializar el *buffer* de frames de vídeo fijándole a cada frame la misma resolución que la del streaming de entrada (*vc->width, vc->height)* en vez de la resolución 32x32 (*SIZE)* píxeles:

```
. . .
initVideoFrame(&(sGC->t_buffers[idb].VideoBuffer[f].frame), vc->pix_fmt, SIZE,
SIZE, sGC);
. . .
```

Código 5-2-1-c. Inicialización del frame (resolución 32x32 píxeles).

```
. . .
initVideoFrame(&(sGC->t_buffers[idb].VideoBuffer[f].frame), vc->pix_fmt, vc-
>width, vc->height, sGC);
. . .
```

Código 5-2-1-d. Inicialización del frame (resolución original).

5.2.2 Cambio en los códecs de salida

5.2.2.1 Salida de vídeo

Debido a que versión anterior del programa guardaba los vídeos en una resolución de 32x32 píxeles, utilizaba la codificación de salida en formato *raw*, que es una compresión sin pérdidas. En este caso, para el vídeo se ha utilizado *MPEG4* como método de compresión. El motivo de este cambio ha sido la reducción en el peso de los vídeos, ya que, al aumentar su resolución ocuparían demasiado espacio si no se comprimen adecuadamente.

Esta modificación se ha realizado cambiando la línea de código que asigna el códec de salida en el fichero sira-tv.h:

```
. . .
#define OUTPUT_CODEC AV_CODEC_ID_RAWVIDEO
. . .
```

Código 5-2-2-a. Códec de vídeo *rawvideo*

Por la línea:

```
. . .
#define OUTPUT_CODEC AV_CODEC_ID_MPEG4
. . .
```

Código 5-2-2-b. Códec de vídeo *MPEG4*

Este cambio conseguirá un gran ahorro de coste computacional y de memoria. En la Figura 19 se hace una comparativa del tamaño que ocupa un spot de 20 segundos de duración utilizando un códec u otro. Se puede observar que con el codificador MPEG-4 el tamaño que ocupa el vídeo es de 5,5MB, y utilizando la codificación planar sin comprimir el tamaño del mismo vídeo es de 318,13 MB.

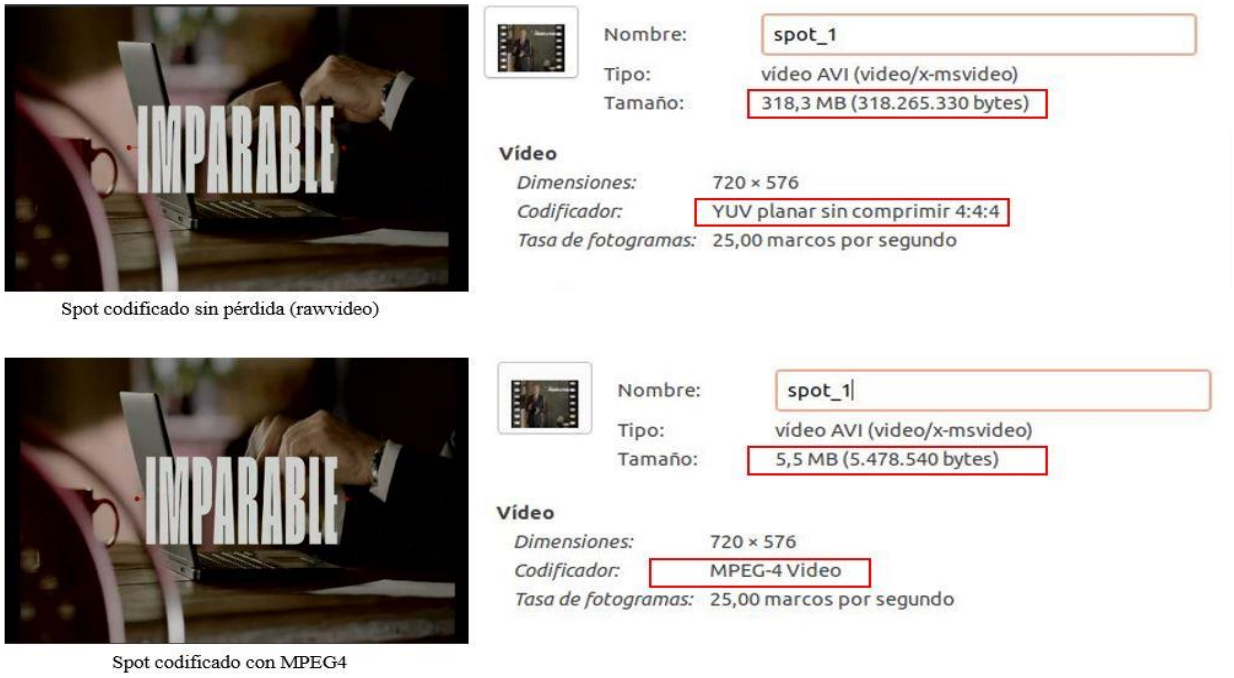

Figura 19. Propiedades de un mismo spot codificado sin pérdidas (arriba) y con MPEG4 (abajo).

5.2.2.2 Salida de audio

Para el caso de audio, se ha sustituido el códec *AC-3 (audio codec)*, por el códec *AAC (advanced audio coding)* que proporciona una salida de audio con más calidad y una mejor tasa de compresión.

Esta modificación se ha realizado añadiendo la línea de código que asigna el códec de salida de audio en el fichero *sira-tv-muxer.c* (función *void saveVideo()*).

```
. . .
out_fmt_ctx->oformat->audio_codec = AV_CODEC_ID_AAC
. . .
```

Código 5-2-3. Códec de audio *AAC*.

5.2.3 Cambio en la detección de los frames negros

La función *int esNegro()* recibe un frame en formato YUV[1] y determina si es un frame negro. Esta tarea la hace recorriendo y evaluando cada píxel del plano Y (luminancia) del frame para calcular número total de píxeles

[1] YUV es un sistema de codificación de colores en el que cada color tiene una señal de Y luminancia (brillo) y dos señales U,V de crominancia (color).

reconocidos como negros. Posteriormente, determina si el frame es negro en función al porcentaje de píxeles negros con respecto al total, el porcentaje mínimo de píxeles no negros que el frame debe superar para no ser considerado negro es del 8%.

Un problema poco común, pero ocurrente en algún tipo de anuncios que acaban con el logo de la marca tras un fondo negro, era una falsa detección de dichos frames como negros. Esto originaba un error a la hora de reconocer el siguiente anuncio. Para solucionar este problema, se ha cambiado la resolución del frame de entrada a la función, pasando de recibir un frame con resolución 32x32 píxeles a un frame con resolución 720x576 píxeles. En las Figuras 20 y 21 se puede observar la diferencia de resoluciones entre los frames que el programa envía de entrada a la función *int estNegro()*. Para el caso de la Figura 20 la función lo reconoce como un frame negro y para el caso de la Figura 21 lo reconoce como no negro.

Figura 20. Frame con resolución 32x32 píxeles detectado como negro por el programa.

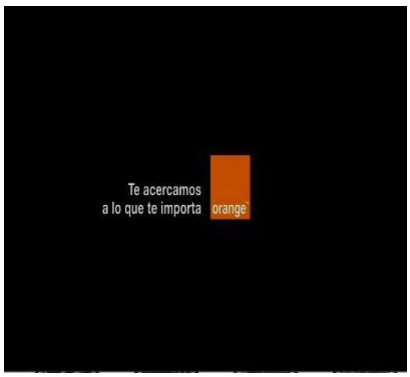

Figura 21. Frame con resolución 720x576 píxeles que el programa no detecta como negro.

5.3 Novedades introducidas en el programa

5.3.1 Elección del formato de salida de los vídeos no reconocidos

Esta nueva opción, con motivo de otorgar al programa una mayor versatilidad, permitirá al usuario especificar el formato en el que se guardarán los vídeos que el programa no sea capaz de reconocer, pudiendo elegir entre "*.mp4*" o "*.avi*". Anteriormente sólo se guardaban en formato "*.avi*".

Utilizando como argumento la opción *-g* o *–guarda* seguida del formato *[avi]* o *[mp4]*, el programa añade la extensión y elige los códecs necesarios para la codificación del vídeo. Esto es muy útil pues los vídeos que no reconozca son posibles candidatos a formar parte de la base de datos en el futuro.

En caso de que el programa no reciba la opción *-g* o *--guarda* a la entrada, no guardará los vídeos en memoria, otra opción muy útil cuando el usuario solo se quiera centrar en la detección de anuncios, no se disponga de memoria suficiente y/o se quiera ahorrar coste computacional en la ejecución del programa, como, por ejemplo, para realizar tareas de depuración del código.

5.3.2 Cambio en la función de extracción de la huella

Como se explica en el capítulo siguiente, se ha introducido una nueva función de hashing que calcula la huella del vídeo utilizando el método "ahash".

La función de hashing tiene que ser la misma en los dos módulos, tanto para crear la base de datos como para extraer la huella del anuncio en directo. A raíz de este cambio se tendrán diferentes bases de datos de anuncios, unas calculadas con la función de hashing de la versión anterior, y otras calculadas con la nueva función de hashing.

5.4 SIRA-TV-2021

Es el nuevo módulo que contiene la versión actualizada de sira-tv-2.1 tras los cambios y actualizaciones descritos en los apartados anteriores.

6 ALGORITMO DE HASHING

Con la premisa de buscar incesantemente la mayor eficiencia para el programa, nos centraremos en esta ocasión en la calidad de la comparación y la detección de los anuncios. El objetivo es claro, conseguir una detección de anuncios rápida, fiable y que por consiguiente, sea robusta ante las distintas distorsiones que pueda sufrir la señal de vídeo de entrada.

Para lograr esta finalidad, se necesitará implementar en el sistema de Video Fingerprinting de la aplicación SIRA-TV el método de hashing que más se adecúe a los requisitos que propone. Como se ha mencionado anteriormente, en la actualidad existen un gran número de métodos de hashing de imágenes y vídeos y todos proponen estrategias y extracción de características totalmente distintas. Elegir un método u otro, va a depender del uso que le vaya a dar el usuario final.

En el caso de la aplicación SIRA-TV, los anuncios publicitarios son de muy corta duración, con lo que extraer rápidamente características de sus primeros frames, resulta la forma más adecuada de obtener las huellas de los vídeos.

Al conjunto de métodos que se encargan de extraer la huella de una imagen utilizando sus características perceptuales se le denomina perceptual hashing, y actualmente hay numerosos algoritmos distintos como son: ahash, phash, dhash, whash, etc. El estudio sobre la calidad y eficiencia de estos algoritmos está recogido en el Trabajo de Fin de Máster de Mónica Pérez Hernández de 2019. [15].

Se ha decidido aplicar el resultado de sus estudios e investigación al programa para ver si se producen mejoras en la detección de los anuncios.

6.1 Algoritmo de hashing basado en la 2D-DCT

Este algoritmo es con el que la versión anterior del programa realizaba la extracción de la huella. Su funcionamiento es el siguiente:

1. Recibe un frame con una resolución de 32x32 píxeles (1024 píxeles en total).
2. Cambia el espacio de color del frame de entrada pasando de RGB a YUV y solo trabajará sobre el plano ‘Y’ (de luminancia).
3. Al plano ‘Y’ del frame de entrada le realiza la Transformada Discreta del Coseno de dos dimensiones y almacena el resultado en una matriz de coeficientes.
4. Evalúa cada término de la matriz de coeficientes haciéndolo igual a ‘0’ si contiene un valor negativo y a ’1’ en otro caso.
5. Almacena el resultado en un vector de 1024 elementos.
6. Reescala el vector en una matriz de 16 filas por 64 columnas que será la huella de la imagen.

Al utilizar la DCT, se pasa la imagen al dominio frecuencial, con lo que para frames con una alta resolución y bien detallados (altas frecuencias) puede ser muy útil este algoritmo, pero para frames con una resolución baja como los frames de 32x32 píxeles que utiliza SIRA-TV para calcular la huella, no es tan útil ya que el contenido frecuencial de la imagen es pobre.

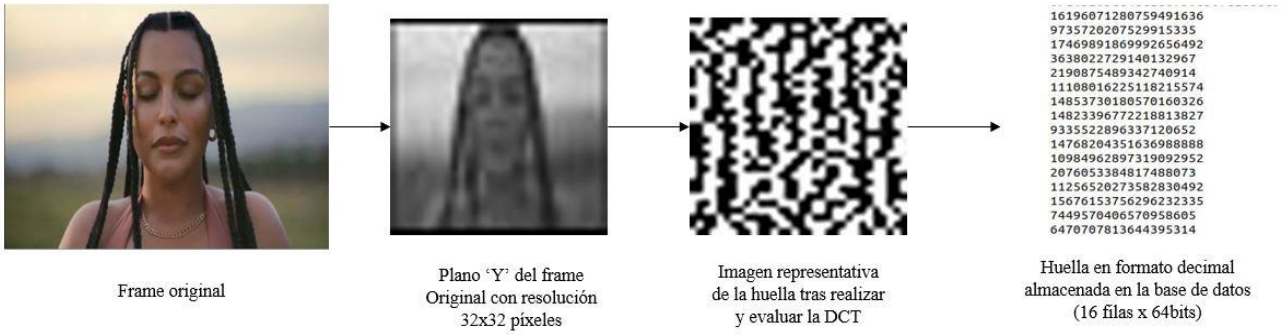

Figura 22. Resultado de aplicar algoritmo de hashing basado en la 2D-DCT al frame de un anuncio.

6.2 Algoritmo Average hashing

6.2.1 Descripción del algoritmo

Average hashing, más conocido como "ahash", es un algoritmo de hashing cuyo fundamento consiste en realizar un promediado de los píxeles de la imagen. Su posible ventaja ante otro algoritmo es que, ante imágenes con bajas frecuencias, es decir, pequeñas y con pocos detalles como las de 32x32 píxeles con las que trabaja el programa, proporciona un hash que representa mucho mejor su estructura no teniendo en cuenta los pequeños detalles.

Por este motivo y tras ver los resultados del trabajo comentado anteriormente [15], se ha implementado una nueva función que se añadirá a los dos módulos del programa para que calcule el hash utilizando este algoritmo. Más adelante se mostrarán los resultados obtenidos tras la evaluación y comparación de este método frente al método que implementa la DCT.

El algoritmo ahash funciona de la siguiente manera:

1. Recibe un frame con una resolución de 32x32 píxeles (1024 píxeles en total).
2. Cambia el espacio de color del frame de entrada pasando de RGB a YUV y solo trabaja sobre el plano 'Y' (de luminancia).
3. Suma todos los valores de luminancia de los 1024 píxeles y calcula el valor medio total.
4. Crea una matriz de coeficientes en la que compara, píxel a píxel, si el valor de la luminancia del píxel es mayor que el valor medio; si es mayor que el valor medio le da un valor '1' a dicho píxel y si no es mayor, '0'.
5. Almacena los valores de la matriz de coeficientes en un vector de 1024 elementos.
6. Reescala el vector en una matriz de 16 filas por 64 columnas que será la huella de la imagen.

Este algoritmo es rápido y tiene poco coste computacional, por lo que parece ser una buena alternativa para la detección de anuncios a tiempo real.

La huella de la imagen permanecerá constante ante variaciones de escalado o de relación de aspecto. Los cambios en el contraste o en el brillo tampoco harán que el hash resultante cambie de forma significativa. Aunque es sensible ante variaciones no lineales entre imágenes, como cambios en el histograma de color o correcciones gamma, ya que se altera la posición del promedio, cambiando así el valor del hash.

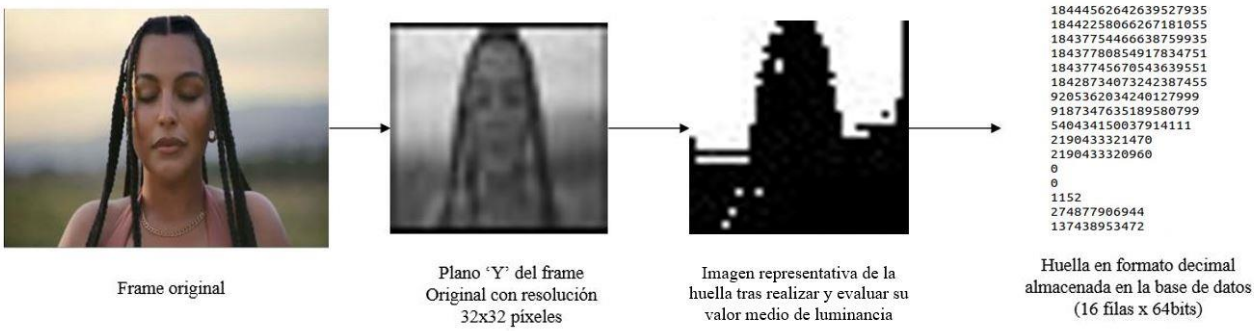

Figura 23. Resultado de aplicar algoritmo de hashing ahash al frame de un anuncio.

6.2.2 Implementación del algoritmo en lenguaje C

Para poder hacer uso del algoritmo ahash en SIRA-TV, se ha implementado en lenguaje C una función alternativa a *void calculaHuella*, llamada *void calculaHuella_aHash*, en *sira-tv-huellas.c* que contiene las siguientes líneas de código:

Código 6-2-2. Función void calculaHuella_aHash().

```
void calculaHuella_aHash(AVFrame *pFrame, huella_t *huella) {
int tam = pFrame->height * pFrame->width;
int x, y, fila, ind = 0;
huella_t z=0;
float suma, media=0;
float aHash[tam];
float matriz[SIZE][SIZE];
float matrizaux[SIZE][SIZE];
huella_t aux = 1;
for (x=0; x<pFrame->height; x++) {
          for(y=0; y<pFrame.>width;y++) {
                    matriz[y][x]=pFrame->data[0][pFrame->linesize[0]*y + x];
                    suma = suma+matriz[y][x];
          }
}

media = suma/tam;
```

```
for (x=0; x<pFrame->height; x++) {
          for(y=0; y<pFrame.>width;y++) {
          matrizaux[x][SIZE-y-1]=matriz[x][y];
          }
}
for (x=0; x<pFrame->height; x++) {
          for(y=0; y<pFrame.>width;y++) {
                    aHash[ind] = matrizaux[x][y];
                    ind++;
          }
}
fila = -1;
for (z=0; z<tam; z++) {
          if (z%64 == 0){
          fila++;
          huella[fila] = 0;
}
          if(aHash[z]>media) {
          huella[fila] += aux<<(BITS_HUELLA_T-1-z%BITS_HUELLA_T);
          }
}
```

7 PRUEBAS Y COMPARACIÓN ENTRE ALGORITMOS DE HASHING

En este capítulo se describe el proceso de examinar el funcionamiento de los dos algoritmos de hashing (descritos en el capítulo 6) implementados en el sistema de Video Fingerprinting de la aplicación SIRA-TV. Para este cometido, se han realizado múltiples ejecuciones de la aplicación utilizando como entrada una señal de vídeo procedente de un streaming real de televisión.

En [16] se muestran distintas formas de alterar la señal de vídeo de la entrada para comprobar la eficacia del algoritmo que el estudio propone. Se ha acordado implementar la mayoría de estos métodos modificando el streaming de entrada, al que se le han añadido distintas distorsiones para simular los posibles inconvenientes que puede sufrir la señal real de televisión y comparar la eficacia entre los dos métodos de hashing.

El objetivo final será determinar el algoritmo de hashing más adecuado para la aplicación.

7.1 Material utilizado para la batería de pruebas

1. Streaming de 3 horas y 15 minutos de duración de la cadena Telecinco emitido el 1 de junio de 2021 a las 21 horas. Este streaming contiene 7 segmentos de anuncios, en los que se emiten más de 100 spots publicitarios distintos.

2. Herramienta FFmpeg (utilizada desde la línea de comandos) para distorsionar el streaming.

3. Base de datos de 125 anuncios con sus huellas extraídas con el método que implementa la DCT y con los spots publicitarios en formato *.avi*. Llamada *BBDD_dct_avi*.

4. Base de datos de 125 anuncios con sus huellas extraídas con el método ahash y con los spots publicitarios en formato *.avi*. Llamada *BBDD_aHash_avi*.

5. Microsoft Excel para organizar, clasificar los resultados y representar las gráficas.

7.2 Metodología de las pruebas

7.2.1 Creación de las bases de datos de anuncios

Para la realización de las pruebas se crearon las dos bases de datos que contienen 125 spots publicitarios en formato *.avi* (ver Figura 24), una con las huellas calculadas con el método que implementa la DCT y otra con las huellas calculadas con el método ahash.

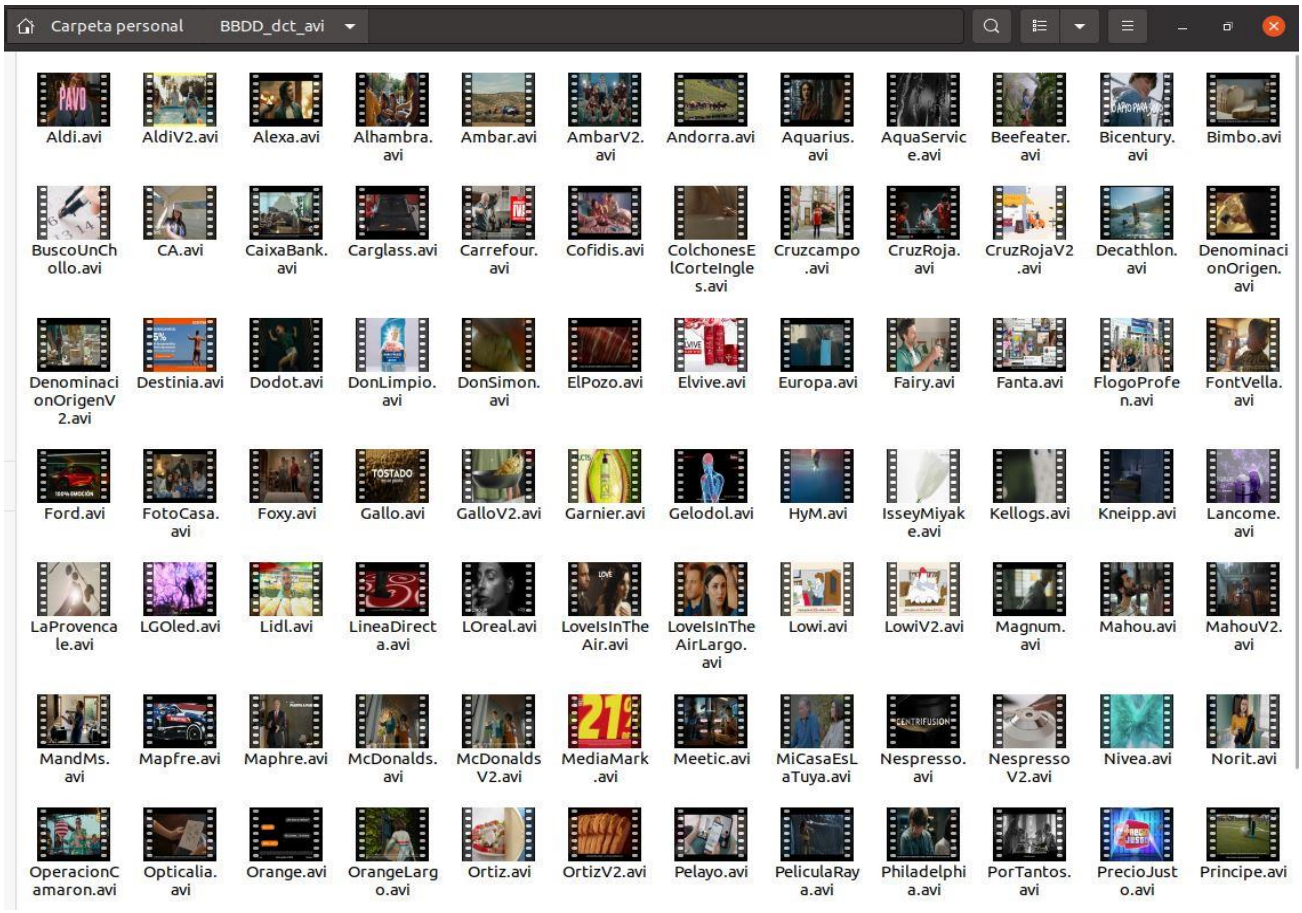

Figura 24. Base de datos de spots publicitarios antes de la ejecución del programa.

Mediante la línea de comandos se ejecuta a la aplicación SIRA-TV-FINGERPRINTER-0.4 que recibe como parámetros de entrada ***-d*** *[directorio de la base de datos]* y ***-f*** *[formato de los vídeos de la base de datos]* y guarda la información de salida en el fichero especificado (Código 7-2-1 y 7-2-2). Tras la ejecución, se crean en la base de datos dos ficheros de texto con las huellas de los vídeos que la conforman (Figura 25). En el Anexo A se muestra la información de salida resultado de realizar esta ejecución.

```
miguel@miguel-VirtualBox:~/workspace/sira/Debug$ ./sira -d /home/miguel/BBDD_dct_avi/ -f .avi > /home/miguel/log_bbdd_dct
```

Código 7-2-1. Ejecución de sira-tv-fingerprinter para crear *BBDD_dct_avi*

```
miguel@miguel-VirtualBox:~/workspace/sira/Debug$ ./sira -d /home/miguel/BBDD_aHash_avi/ -f .avi >
/home/miguel/log_bbdd_aHash
```

Código 7-2-2. Ejecución de sira-tv-fingerprinter para crear *BBDD_aHash_avi*

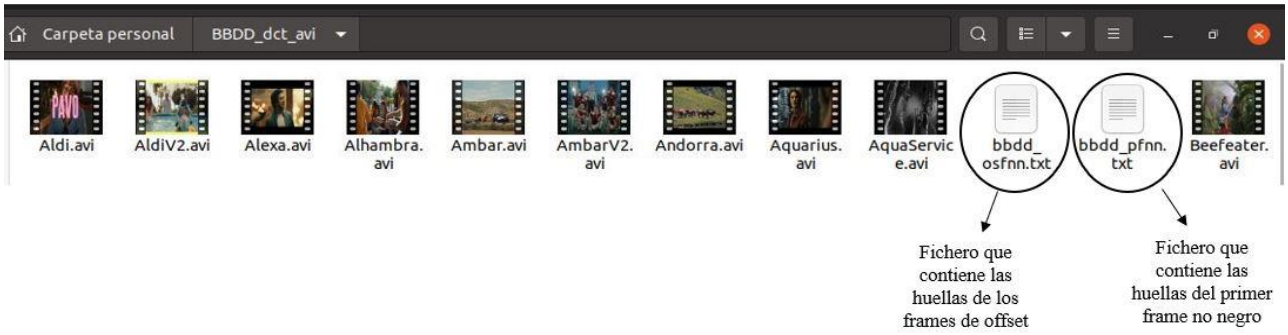

Figura 25. Base de datos tras la ejecución de SIRA-TV-FINGERPRINTER-0.4.

7.2.2 Ejecución de la aplicación SIRA-TV-2021

Con las bases de datos creadas se realizaron diversas ejecuciones de la aplicación SIRA-TV-2021 sobre el streaming de entrada.

Utilizando la consola de Ubuntu, se ejecuta la aplicación SIRA-TV-2021 con la orden:

```
miguel@miguel-VirtualBox:~/workspace/sira-tv-2021/Debug$ ./sira-tv-2021 -c TELECINCO -i 5 -s
/home/miguel/Streaming_T5/telecinco.mp4 -d /home/miguel/BBDD_aHash_avi/ -f .avi -g avi >
/home/miguel/log_ejecucion_aHash
```

Código 7-2-3. Ejecución de sira-tv-2021

que recibe como parámetros de entrada: ***-c*** *[nombre del canal]* ***-i*** *[identificador del canal]* ***-s*** *[ruta del streaming de entrada]* ***-d*** *[ruta de la base de datos]* ***-f*** *[formato de los vídeos de la base de datos]* ***-g*** *[formato de salida de los vídeos no reconocidos]*.

La información de salida se guarda en el fichero */home/miguel/log_ejecución_dct.txt* que se muestra en el Anexo B.

Para cada versión distinta del streaming de entrada se evaluaron por separado los dos métodos de hashing y se varió el Umbral Hamming para cada método. El Umbral Hamming se modifica cambiando la línea de código de *sira-tv.h* en dónde se define la distancia máxima para coincidencia de huellas:

```
#define UMBRAL_HAMMING (1024*[valor_en_%_del_umbral]/100)
```

En la siguiente tabla se muestran los valores que ha recibido este parámetro.

Tabla 7-1. Umbral Hamming utilizado para las pruebas del algoritmo de hashing.

Umbral Hamming (%)	Distancia Hamming máxima para coincidencia de huellas
1	10 bits
3	30 bits
5	51 bits
10	102 bits
18	184 bits

Todos los resultados obtenidos serán en función del Umbral Hamming (al que se le llamará también umbral de decisión). Si un algoritmo necesita menos Umbral Hamming para funcionar correctamente, más se parecen las huellas calculadas a los vídeos de la entrada del algoritmo con las que guarda en la base de datos. Cuanto más grande sea el Umbral Hamming mayor grado de tolerancia habrá para considerar vídeos distintos como iguales.

7.2.3 Modificación del streaming de entrada para añadirle distorsiones

Para realizar las distorsiones, se hacen uso de los filtros *vf_filter* contenidos en la librería *libavfilter* de FFmpeg. Como explica su web oficial [17], para utilizar estos filtros sobre un streaming de entrada se emplea una orden con formato:

./ffmpeg ***-i*** *[streaming_de_entrada]* ***-vf*** *[filtro]-**c**:a copy [straming_de_salida]*

Se han implementado los filtros que se muestran a continuación:

Tabla 7-2. Filtros de FFmpeg utilizados para añadir distorsiones al streaming de entrada.

Filtro	Función
fps	Cambia la tasa de fotogramas (*frames por segundo*)
noise	Añade ruido al streaming de entrada
gblurr	Añade el filtro 'Gaussian blur' al streaming de entrada
eq	Cambia la correción gamma del streaming de entrada

7.2.4 Resultados posibles tras la ejecución del programa

Cuando se detecta un vídeo y se realiza su búsqueda con la base de datos se abarcan cuatro posibles escenarios:

1- Verdaderos Positivos (**VP**): Dos anuncios iguales se han considerado como iguales.
2- Verdaderos Negativos (**VN**): Dos anuncios distintos se han considerado como distintos.
3- Falsos Positivos (**FP**): Dos anuncios distintos se han considerado como iguales.
4- Falsos Negativos (**FN**): Dos anuncios iguales se han considerado como distintos.

Estos escenarios son los que se utilizarán para valorar la eficacia de un algoritmo de hashing frente al otro. Un algoritmo será más eficaz cuando maximice el número de Verdaderos Positivos y minimice el número de Falsos Positivos.

La probabilidad de estos sucesos se ha obtenido en función del número total de cambios en la programación (vídeos a la entrada del algoritmo de hashing) y del número total de anuncios emitidos durante el streaming.

1- Probabilidad de Verdadero Positivo ($\mathbf{P_{VP}}$): número de Verdaderos Positivos (**VP**) frente a número de anuncios emitidos.
2- Probabilidad de Falso Positivo ($\mathbf{P_{FP}}$) número de Falsos Positivos (**FP**) frente a número total de cambios en la programación.

7.3 Realización de las pruebas

En primer lugar, se ha analizado manualmente el streaming completo y se han anotado por orden los anuncios emitidos durante la emisión: el número total es de 159 fragmentos de programación destinada a anuncios en los cuales hay 125 spots publicitarios distintos.

En el caso ideal (el algoritmo de hashing no produce ningún fallo), la salida del programa debe mostrar que se han detectado 174 cambios en la programación en los cuales se han encontrado coincidencias con la base de datos en 159 anuncios, no se han encontrado coincidencias en 6 anuncios y se han detectado 9 segmentos de programación (Figura 26). Tras observar los segmentos de vídeo correspondientes a los 15 segmentos de vídeo sin encontrar coincidencias se descartaron, pues no corresponden a anuncios publicitarios, y por lo tanto no están en la base de datos. Por consiguiente, se obtiene un 100% de detección.

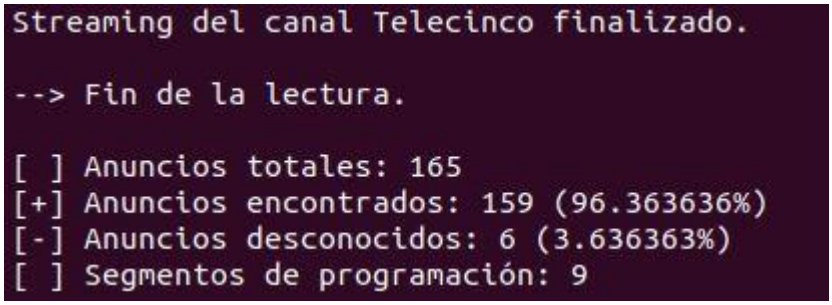

Figura 26. Información de salida tras la ejecución de sira-tv-2021

Una vez conocida la salida ideal del streaming se ha ejecutado la aplicación SIRA-TV-2021 utilizando un método de hashing distinto en cada caso, sobre cada streaming modificado, variando el Umbral Hamming. Los datos de salida obtenidos tras cada ejecución se han anotado en Excel para obtener los resultados y están reflejados en el Anexo C.

7.3.1 Pruebas sobre un streaming sin distorsiones

Los datos obtenidos indican que, a medida que aumenta el umbral de decisión (Umbral Hamming), el método ahash funciona correctamente y alcanza el 100% de verdaderos positivos antes que el algoritmo que implementa la DCT. Si se sigue aumentando el umbral de decisión para que el método que implementa la DCT obtenga el 100% de verdaderos positivos, el algoritmo ahash obtendrá un número reducido de falsos positivos, esto se debe a que el valor de la distancia de Hamming máxima entre vídeos distintos se hace cada vez mayor, con lo que la diferenciación en los vídeos es más complicada.

El siguiente gráfico muestra la probabilidad de los sucesos anteriores (P_{VP} y P_{FP}) de cada algoritmo en función del Umbral Hamming empleado para la detección.

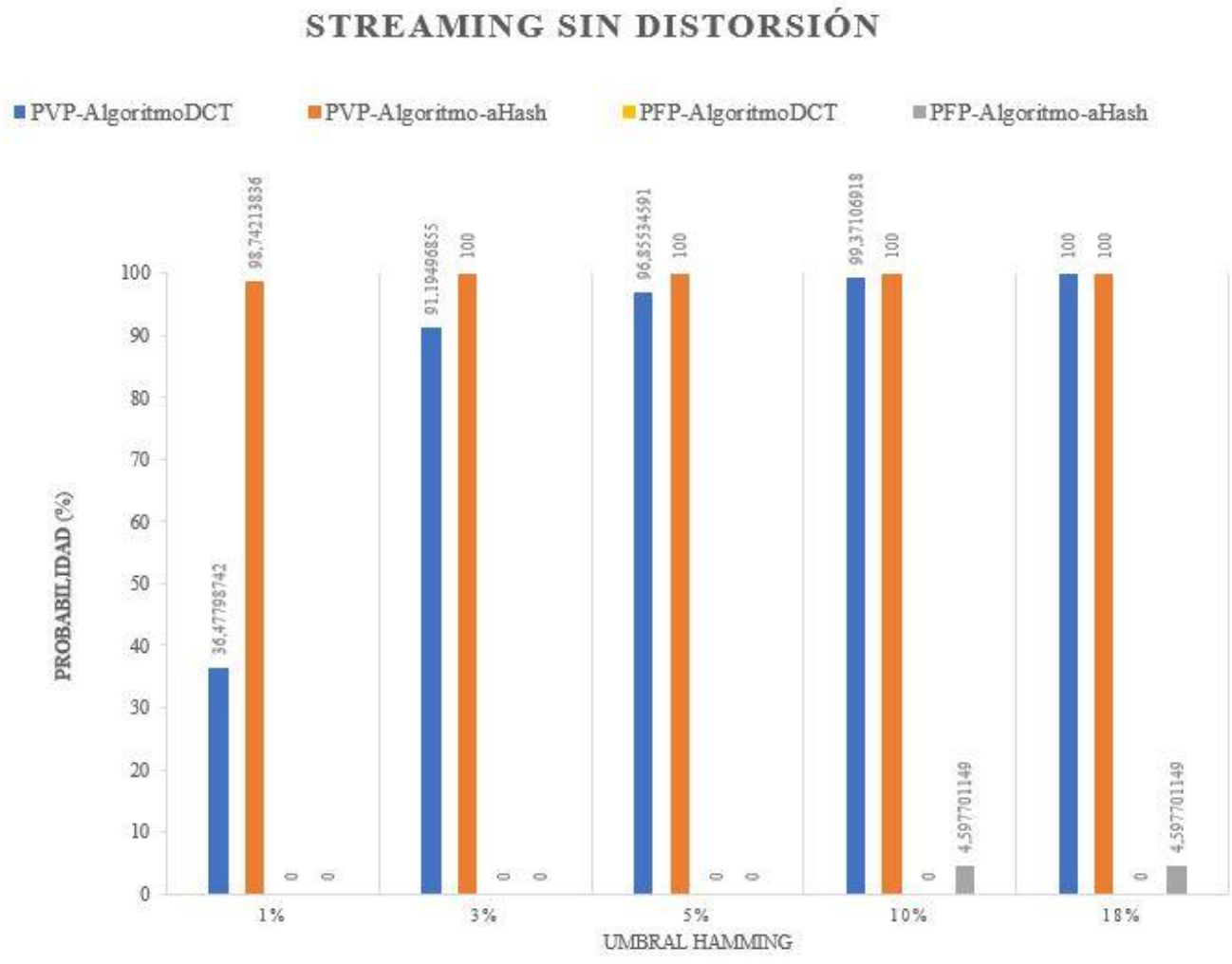

Gráfico 7-1. P_{VP} y P_{FP} de cada algoritmo para cada Umbral Hamming. (streaming sin distorsión)

7.3.2 Pruebas sobre un streaming con disminución de FPS

La velocidad a la que se suceden los fotogramas en un vídeo está determinada por los FPS (*frames per second)*. Se ha probado a disminuir esta velocidad pasando de los 25 frames por segundo a los 15. En este caso, es muy probable que el programa elija un frame distinto al de la base de datos para calcularle la huella, pues por cada segundo de vídeo se han eliminado 10 frames, con lo que calculará la huella unos frames después del primer frame no negro de cada anuncio.

Para reducir la tasa de fotogramas por segundo se ha hecho uso del filtro *vf_fps* de FFmpeg con la siguiente orden por línea de comandos:

```
miguel@miguel-VirtualBox:~/git/ffmpeg$ ./ffmpeg -i /home/miguel/Streaming_T5/telecinco_n.mp4 -vf
fps=15 -c:a copy /home/miguel/Streaming_T5/15fps/telecinco_n.mp4
```

Código 7-2-1. Comando de FFmpeg para reducir la tasa de fotogramas por segundo.

Los resultados indican que en ninguno de los dos casos se consigue un 100% de verdaderos positivos. Esto significa que, para determinados anuncios, los cambios entre frames sucesivos son tan bruscos que hacen que sus huellas sean muy distintas. Aun así, el algoritmo ahash se acerca a valores muy próximos al 100% de verdaderos positivos a medida que aumenta el umbral de decisión. Con respecto al número de falsos negativos, en ambos casos es similar, reconociendo un porcentaje muy pequeño de vídeos distintos como iguales.

El siguiente gráfico muestra la probabilidad de los sucesos anteriores (P_{VP} y P_{FP}) de cada algoritmo en función del Umbral Hamming empleado para la detección.

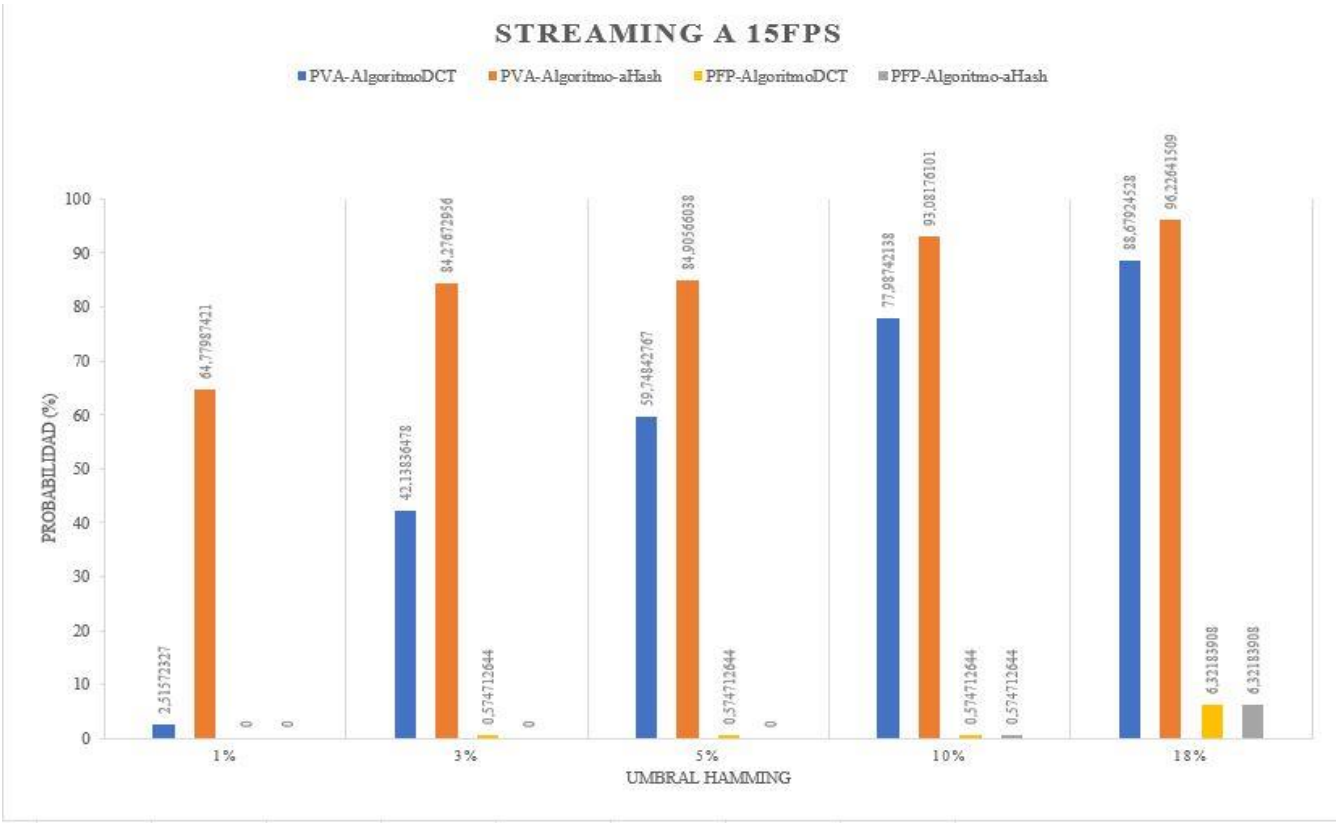

Gráfico 7-2. P_{VP} y P_{FP} de cada algoritmo para cada Umbral Hamming (streaming con reducción de fps)

7.3.3 Pruebas sobre un streaming con Ruido Blanco Gaussiano

El ruido blanco gaussiano aditivo suele ser la forma típica de modelar el ruido. En este caso, a la señal de vídeo del streaming se le superpone dicho ruido, la potencia viene determinada por su desviación típica.

Para generar este ruido, se ha utilizado la definición de ruido blanco gaussiano que se muestra en [18] y se ha comprobado su relación con las líneas de código del filtro *vf_noise.c* de FFmpeg. La desviación típica del ruido (σ) se modelará en función del parámetro *'strength'* que reciba el filtro a la entrada, dividiendo dicho valor por un factor igual a raíz cuadrada de tres.

$$\sigma = \frac{strength}{\sqrt{3}}$$

El parámetro *'strength'* debe ser un número entero comprendido entre 0 y 100. El caso límite de este proceso es en el que el programa deja de detectar correctamente todos los cambios en la programación, es decir, cuando los frames negros ya no sean detectados como tal. Para situarnos en dicho caso hay que asignar al parámetro '*strength'* un valor igual a 18, con lo que la desviación típica (σ) del ruido gaussiano será igual a 10'39.

Además, si se quiere que el patrón del ruido cambie con el paso de los frames, hay que añadir la opción 't' a la entrada del filtro.

Utilizando FFmpeg desde la línea de comandos con la orden:

```
miguel@miguel-VirtualBox:~/git/ffmpeg$ ./ffmpeg -i /home/miguel/Streaming_T5/telecinco.mp4 -vf
noise:alls=18:allf=t -c:a copy /home/miguel/Streaming_T5/AWGN/telecinco.mp4
```

Código 7-2-3. Comando de FFmpeg para añadir Ruido Gaussiano Blanco a un streaming.

se añade ruido blanco gaussiano, de desviación típica (σ) igual a 10'39, al vídeo de la entrada y se almacena el vídeo resultante en la ruta de salida. El resultado visual de este proceso se muestra en la Figura 27.

Figura 27. Streaming antes y después de añadirle Ruido Blanco Gaussiano.

En comparación con un streaming sin ruido, se obtienen muchos más errores en la detección en ambos métodos,

aunque para umbrales de decisión pequeños (1%, 3%, 5%) el método ahash obtiene un porcentaje de verdaderos positivos mucho mayor que el método que implementa la DCT. Cuando el Umbral Hamming aumenta, el método que implementa la DCT aumenta su número de verdaderos positivos y el método de ahash produce un número muy reducido de falsos positivos.

El siguiente gráfico muestra la probabilidad de los sucesos anteriores (P_{VP} y P_{FP}) de cada algoritmo en función del Umbral Hamming empleado para la detección.

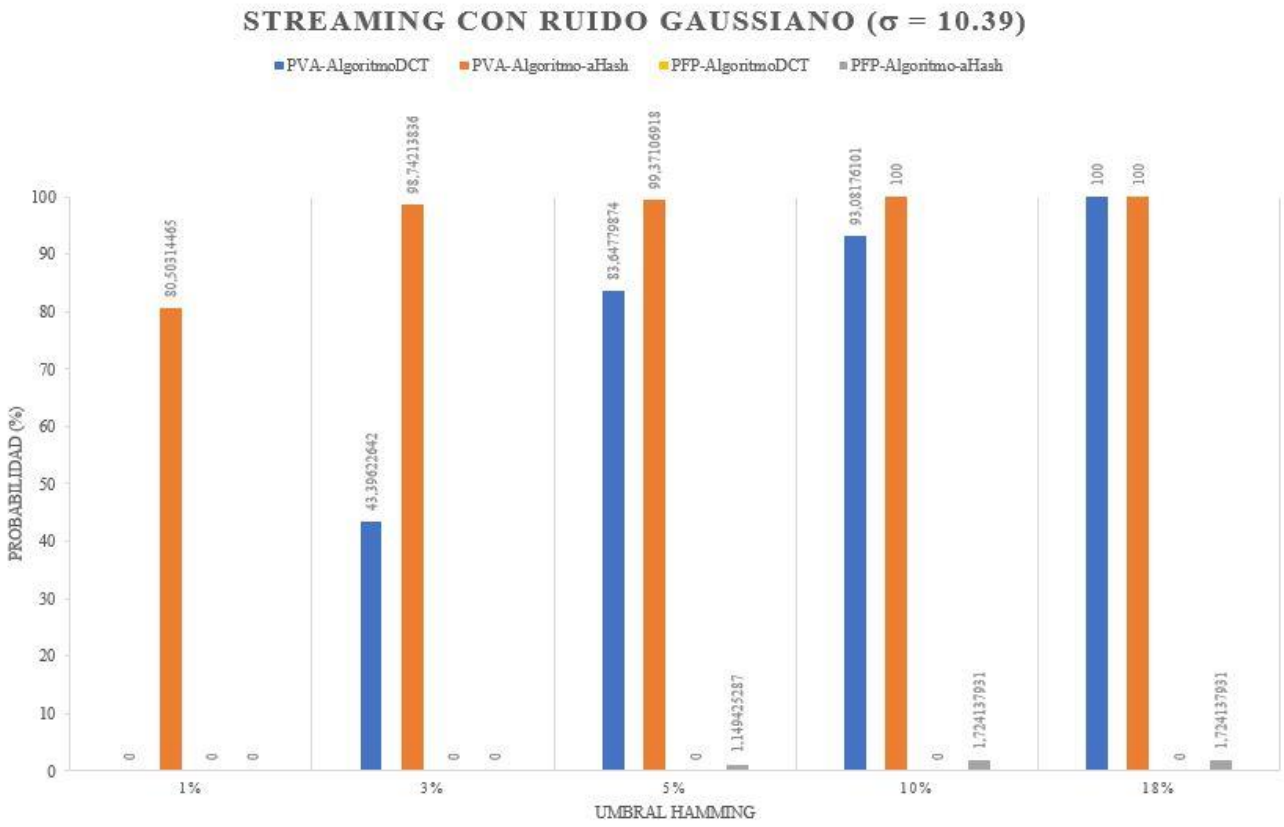

Gráfico 7-3. P_{VP} y P_{FP} de cada algoritmo para cada Umbral Hamming (streaming con ruido gaussiano)

7.3.4 Pruebas sobre un streaming con el filtro de difuminado Gaussian blur

El filtro Gaussian blur, aplicado a un vídeo, se utiliza para producir un efecto de difuminado sobre sus frames. El filtro usa dos funciones matemáticas, una para el eje vertical y otra para el eje horizontal, y el número de píxeles que va a utilizar por cada eje lo recibe como parámetro (‘*ratio’*). Con estas dos funciones determina una tercera función que crea una distribución normal (de desviación típica recibida como parámetro) de los valores de estos píxeles.

Para implementar este filtro, se ha utilizado el filtro *vf_gblur* de FFmpeg mediante la siguiente orden por línea de comandos:

```
miguel@miguel-VirtualBox:~/git/ffmpeg$ ./ffmpeg -i /home/miguel/Streaming_T5/telecinco.mp4 -vf
gblur=sigma=5 -c:a copy /home/miguel/Streaming_T5/gblur/telecinco.mp4
```

Código 7-2-4. Comando de FFmpeg para utilizar el filtro Gaussian blur sobre un streaming

El filtro que se añade al streaming de entrada tras esta orden tendrá un radio de 1 píxel (valor por defecto) y una desviación típica igual a 5. El resultado se puede apreciar visualmente en la Figura 28.

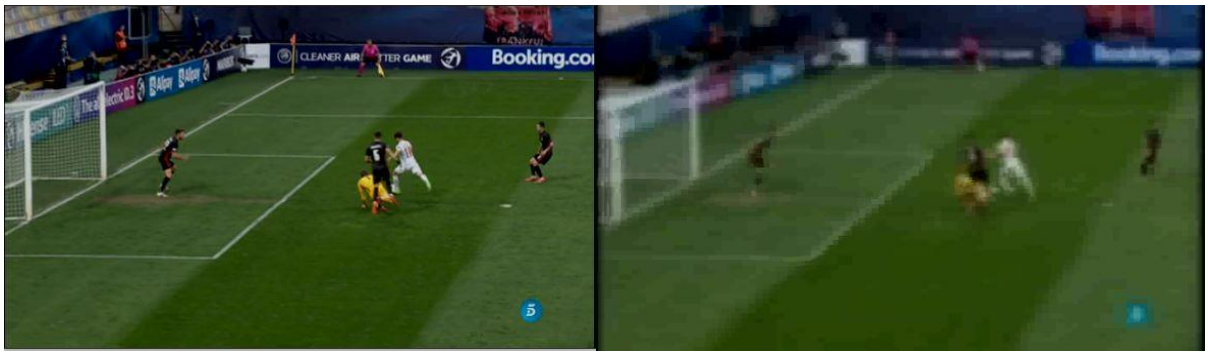

Figura 28. Frame de un vídeo antes y después de aplicar el filtro Gaussian blur.

Los resultados indican que, ante un streaming de entrada con efecto de difuminado, el algoritmo ahash necesita menos distancia Hamming entre vídeos para reconocer correctamente los anuncios. Al aumentar el umbral de decisión, cuando el algoritmo que utiliza la DCT consigue detectar el 100% de los anuncios, el algoritmo ahash empieza a confundir un número reducido de anuncios no identificados con anuncios de la base de datos (falsos positivos).

El siguiente gráfico muestra la probabilidad de los sucesos anteriores (P_{VP} y P_{FP}) de cada algoritmo en función del Umbral Hamming empleado para la detección.

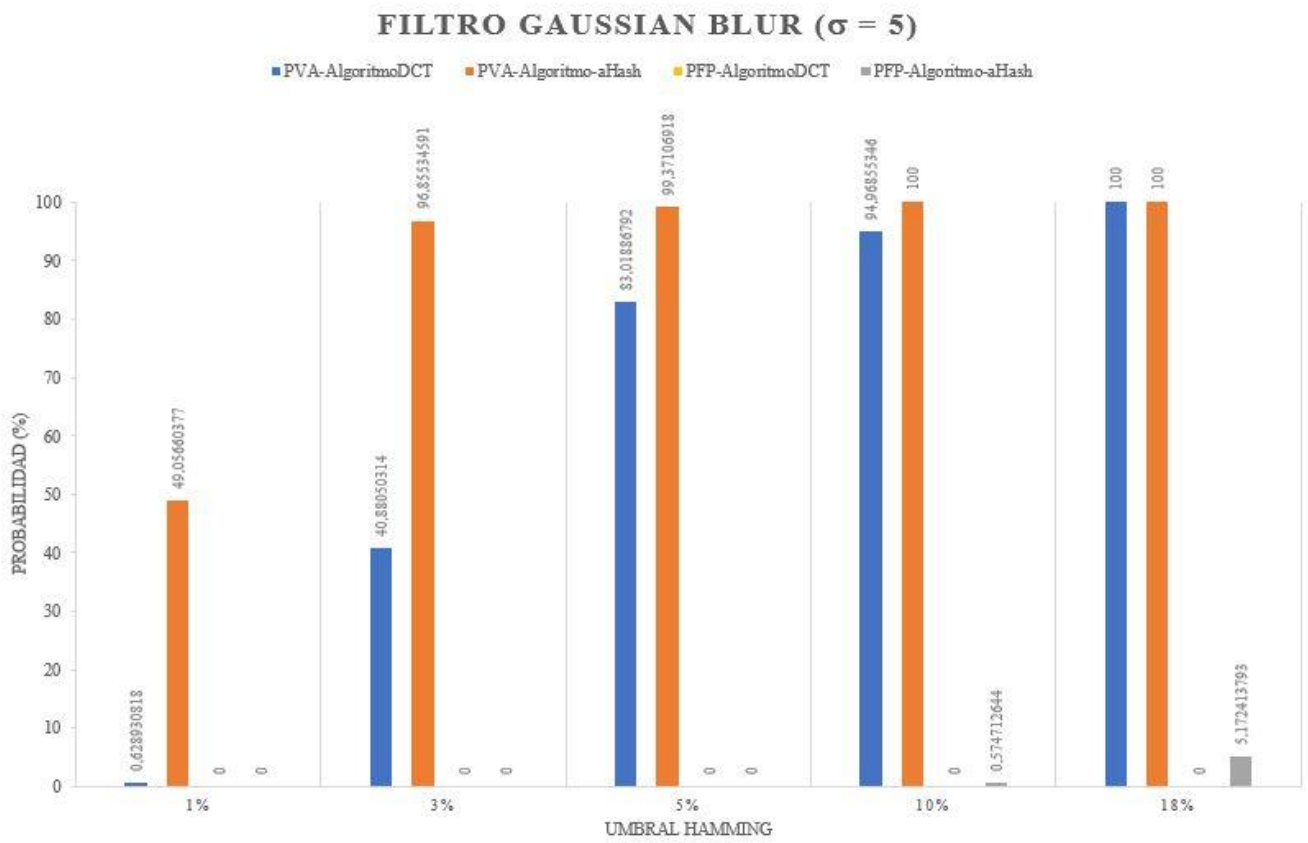

Gráfico 7-4. P_{VP} y P_{FP} de cada algoritmo para cada Umbral Hamming (streaming con filtro Gaussian blur)

7.3.5 Pruebas sobre un streaming con un aumento en la corrección gamma (γ) del 30%

La corrección gamma es una operación no lineal que se utiliza para codificar y decodificar valores de luminancia en sistemas de vídeo. Se puede definir mediante la siguiente fórmula:

$$V_{OUT} = AV_{in}^{\gamma}$$

Se le ha aplicado al streaming de la entrada un aumento en la corrección gamma del 30% utilizando la orden de FFmpeg:

```
miguel@miguel-VirtualBox:~/git/ffmpeg$ ./ffmpeg -i /home/miguel/Streaming_T5/telecinco.mp4 -vf
eq=gamma=1.3 -c:a copy /home/miguel/Streaming_T5/CGamma/telecinco.mp4
```

Código 7-2-5. Comando de FFmpeg para aumentar la corrección gamma del streaming de entrada en un 30%

En la Figura 29 se pueden ver los efectos de este aumento en la corrección gamma.

Figura 29. Streaming antes y después de un aumento en la corrección gamma γ (+30%)

Los resultados indican que a medida que el umbral de decisión aumenta, lo hace también el número de verdaderos positivos siendo el incremento mucho mayor en el algoritmo ahash. En los dos casos, cuando el Umbral Hamming se fija en un 18% existen algunos casos de falsos positivos.

El Gráfico 7-5 muestra la probabilidad de los sucesos anteriores (P_{VP} y P_{FP}) de cada algoritmo en función del Umbral Hamming empleado para la detección.

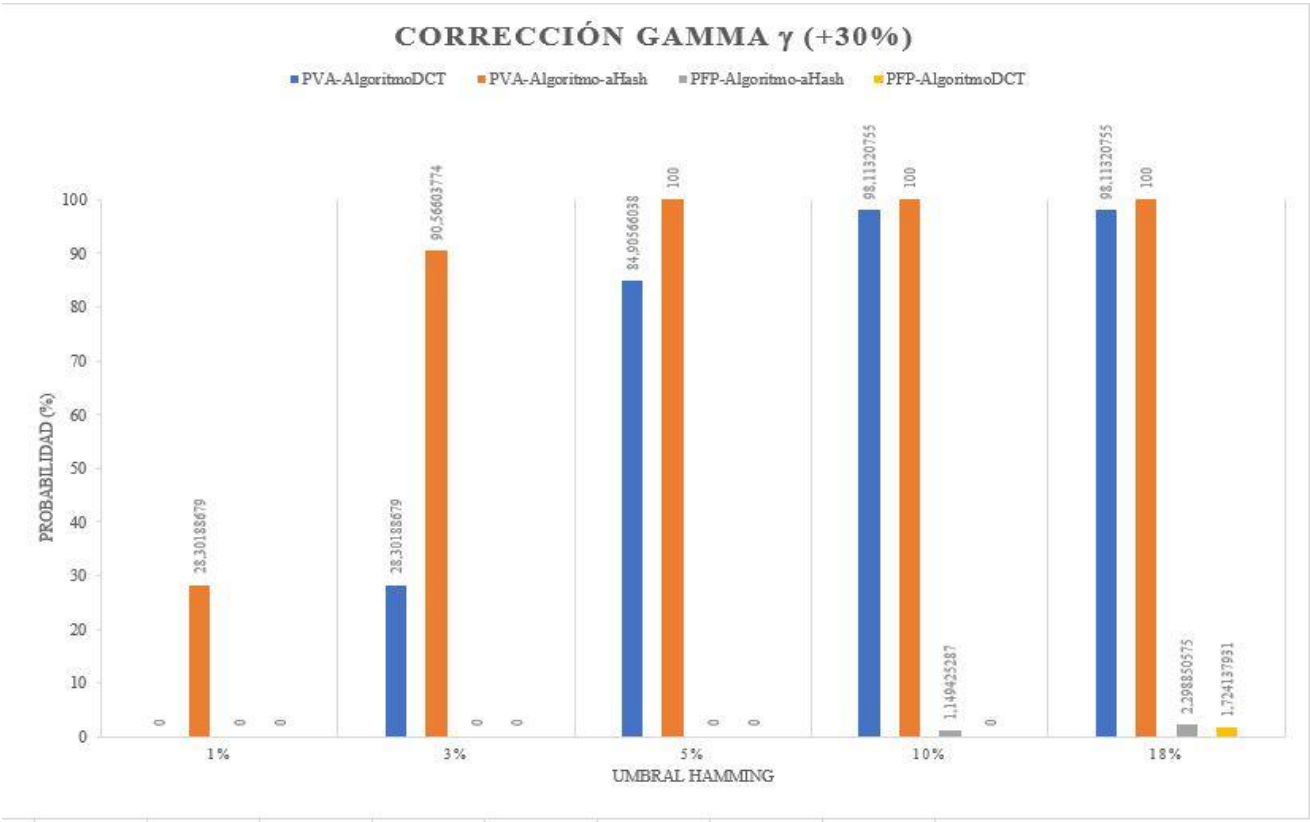

Gráfico 7-5. P_{VP} y P_{FP} de cada algoritmo para cada Umbral Hamming (streaming con aumento en la corrección gamma).

7.4 Valoración de los resultados

La siguiente tabla reúne, para cada algoritmo, la distancia de Hamming mínima a la que se consigue el máximo número posible de Verdaderos Positivos.

Tabla 7-4. Número máximo de bits de diferencia entre huellas para conseguir el mayor número de Verdaderos Positivos.

Tipo de streaming de vídeo a la entrada	Algoritmo ahash	Algoritmo DCT
Original	30 bits	184 bits
Con reducción de fps	185 bits	184 bits
Con ruido gaussiano	102 bits	184 bits
Con filtro de difuminado Gaussian blur	102 bits	184 bits
Con aumento en la corrección gamma	51 bits	184 bits

Todos los resultados indican que el algoritmo ahash necesita una distancia de Hamming menor para funcionar correctamente. Las ventajas de que la distancia de Hamming entre huellas sea menor son las siguientes:

- Mayor robustez de las huellas: las huellas de los anuncios detectados son muy parecidas (en algunos casos idénticas) a las originales.
- Ahorro en coste computacional: en la mayoría de casos no se necesitará realizar una segunda búsqueda en la base de datos utilizando el frame de offset.

Por otra parte, el algoritmo ahash responde mucho mejor ante todas las distorsiones del streaming de entrada.

A la vista de estos resultados, se ha elegido el algoritmo ahash como método principal de extracción de la huella del sistema de Video Fingerprinting de la aplicación SIRA-TV.

8 CONCLUSIÓN Y POSIBLES LÍNEAS DE FUTURO

Tras haber tomado contacto por primera vez con las librerías de FFmpeg, me he dado cuenta de la infinidad de utilidades que tiene esta plataforma, siendo un objeto de estudio muy amplio. El hecho de que la aplicación SIRA-TV se valga de estas librerías no me deja lugar a dudas de que estará siempre provista con las mejores técnicas para la manipulación de contenido multimedia. También me hace suponer que en el futuro necesitará ser actualizada nuevamente, ya que al igual que las tecnologías multimedia, FFmpeg se encuentra en constante desarrollo.

Con respecto al cambio en el algoritmo de hashing de la aplicación, los resultados obtenidos concuerdan a la perfección con los realizados en el estudio [15] en el que nos basamos al principio del proyecto. El algoritmo ahash funciona con mucha precisión necesitando una distancia de Hamming muy reducida para obtener los candidatos. Esto optimizará el programa consiguiendo que las búsquedas en la base de datos sean más efectivas, necesitando solo el primer frame en la mayoría de ocasiones. También responde positivamente ante las distintas distorsiones a las que se ha sometido el streaming. Un único inconveniente que se observa es que el algoritmo ahash obtiene un número muy pequeño de falsos negativos cuando la distancia de Hamming no es excesivamente grande. El caso ideal sería que no confundiera estos segmentos de vídeo. Se propone seguir investigando en nuevos algoritmos y métodos para conseguir llevar a cero este número de falsas detecciones.

Aparte de mantener Sistema Inteligente de Reconocimiento de Anuncios de Televisión actualizado a la última versión de FFmpeg y de seguir investigando en nuevas técnicas para el hashing de vídeos. Sería interesante diseñar una interfaz amigable para la gestión y visualización del contenido multimedia, así como para la información de salida de la aplicación.

REFERENCIAS

[1] https://www.barloventocomunicacion.es/wp-content/uploads/2021/08/EL-ROSCO-del-consumo-audiovisual_Julio-2021_Barlovento-Comunicacion.pdf

[2] Zhu Liu, Tao Liu, David Gibbon, Behzad Shahraray «Effective and Scalable Video Copy Detection» p. 1, 2010.

[3] Xiushan Nie, Xin Zhou, Yang Shi, Jiande Sun, Yilong Yin «Classification-enhancement deep hashing for large-scale video retrieval » *Applied Soft Computing* p. 9, 2021.

[4] Yingxin Wang, Xiushan Nie, Xin Zhou, Yang Shi Yilong Yin «Attention-Based Video Hashing for Large-Scale Video Retrieval» *IEEE Transactions on Cognitive and Developmental Systems* p. 9, 2021.

[5] Ziqing Huang, Shiguang Liu «Perceptual Hashing With Visual Content Understanding for Reduced-Reference Screen Content Image Quality» *IEEE Transactions on Circuits and Systems for Video Technology* p. 9, 2020.

[6] Hannes Mareen, Niels Van Kets, Peter Lambert, Glenn Van Wallendael «Fast Fallback Watermark Detection Using Perceptual Hashesl» *Electronics Recent Developments and Applications of Image Watermarking* p. 5, 2021.

[7] Ching-Yung Lin, Shih-Fu Chang «A robust image authentication method distinguishing JPEG compression from malicious manipulation» *IEEE Transactions on Circuits and Systems for Video Technology* p. 2, 2001.

[8] Julien Law-to, Li Chen, Alexis Joly, Ivan Laptev, Oliver Buisson «Video Copy Detection: a comparative study » *Proceedings of the 6th ACM International Conference on Image and Video Retrieval* p. 7, 2007.

[9] Arun Hampapur, Kiho Hyun, Ruud M.Boile «Comparison of sequence matching techniques for video copy detection » *Proceedings of SPIE - The International Society for Optical Engineering* p. 1, 2002.

[10] Xiushan Nie, Xin Zhou, Yang Shi, Jiande Sun, Yilong Yin «A Visual Model-Based Perceptual Image Hash for Content Authentication » *IEEE Transactions on Information Forensics and Security* p. 7, 2015.

[11] R. Padmashri, S. Srinivasulu, J. R. Raj, J. Jabez, S. Gowri «Perceptual Image Hashing Using Surffor Feature Extraction and Ensemble Classifier » *IEEE 3rd International Conference on Signal Processing and Communication* p. 5, 2021.

[12] (20) Jefatura del Estado. Ley 7/2010, de 31 de marzo, General de la Comunicación Audiovisual. BOE núm. 79, 2010. Sec. I. Disposiciones generales, 5292 p. 30157-209.

[13] José Javier Alcántara Armenteros «Sistema Inteligente de Reconocimiento de Anuncios de Televisión» *Trabajo de Fin de Grado GITT, Universidad de Sevilla* p. 2016

[14] http://ffmpeg.org/doxygen/trunk/

[15] Mónica Pérez Hernandez «Comparación de algoritmos de hashing para la identificación de anuncios en TV» *Trabajo de Fin de Máster IT, Universidad de Sevilla* p. 2019

[16] Sunil Lee, Chang D. Yoo «Robus Video Fingerprinting for Content-Based Video Identification» *IEEE Transactions on Circuits and System for Video Technology* p. 7, 2008

[17] https://ffmpeg.org/ffmpeg-filters.html

[18] M. C. Jeruchim, P. Balaban, K. S. Shanmugan «Simulation of Communication System»

ANEXO A: SALIDA TRAS LA EJECUCIÓN DE SIRA-FINGERPRINTER-0.4

```
Directorios de entrada:   /home/miguel/BBDD_aHash_avi/
Formatos de entrada:   .avi

--> Analizando directorio:   /home/miguel/BBDD_aHash_avi/
      [ ] Omitiendo fichero: '.'
      [ ] Omitiendo fichero: '..'

      [] Cargando vídeo 1: 'T5TransiccionV2.avi'...
      [] Cargando vídeo 2: 'Cofidis.avi'...
      [] Cargando vídeo 3: 'MiCasaEsLaTuya.avi'...
      [] Cargando vídeo 4: 'Meetic.avi'...
      [] Cargando vídeo 5: 'McDonaldsV2.avi'...
      [] Cargando vídeo 6: 'PromoT5EuroV2.avi'...
      [] Cargando vídeo 7: 'VodafoneV2.avi'...
      [] Cargando vídeo 8: 'Magnum.avi'...
      [] Cargando vídeo 9: 'T5PromoEuro4.avi'...
      [] Cargando vídeo 10: 'T5Cine.avi'...
      [] Cargando vídeo 11: 'SunBites.avi'...
      [] Cargando vídeo 12: 'OrtizV2.avi'...
      [] Cargando vídeo 13: 'Andorra.avi'...
      [] Cargando vídeo 14: 'Starbucks.avi'...
      [] Cargando vídeo 15: 'DenominacionOrigenV2.avi'...
      [] Cargando vídeo 16: 'PromoT5Euro.avi'...
      [] Cargando vídeo 17: 'Ortiz.avi'...
      [] Cargando vídeo 18: 'Somat.avi'...
      [] Cargando vídeo 19: 'Orange.avi'...
      [] Cargando vídeo 20: 'Bicentury.avi'...
      [] Cargando vídeo 21: 'Carglass.avi'...
      [ ] Omitiendo fichero: 'bbdd_osfnn.txt'
      [] Cargando vídeo 22: 'Beefeater.avi'...
      [] Cargando vídeo 23: 'Garnier.avi'...
      [] Cargando vídeo 24: 'T5LoveIsInTheAir4.avi'...
      [] Cargando vídeo 25: 'Yoigo.avi'...
      [] Cargando vídeo 26: 'MandMs.avi'...
      [ ] Omitiendo fichero: 'bbdd_pfnn.txt'
```

```
[] Cargando vídeo 27: 'ElPozo.avi'...
[] Cargando vídeo 28: 'LOreal.avi'...
[] Cargando vídeo 29: 'Puleva.avi'...
[] Cargando vídeo 30: 'ColchonesElCorteIngles.avi'...
[] Cargando vídeo 31: 'LoveIsInTheAir.avi'...
[] Cargando vídeo 32: 'Gallo.avi'...
[] Cargando vídeo 33: 'PrecioJusto.avi'...
[] Cargando vídeo 34: 'Norit.avi'...
[] Cargando vídeo 35: 'DonSimon.avi'...
[] Cargando vídeo 36: 'LowiV2.avi'...
[] Cargando vídeo 37: 'Alhambra.avi'...
[] Cargando vídeo 38: 'OrangeLargo.avi'...
[] Cargando vídeo 39: 'T5PromoEuroV3.avi'...
[] Cargando vídeo 40: 'T5PromocionRocioLargo.avi'...
[] Cargando vídeo 41: 'T5Rocio3.avi'...
[] Cargando vídeo 42: 'Nespresso.avi'...
[] Cargando vídeo 43: 'T5PromoSupervivientes.avi'...
[] Cargando vídeo 44: 'MahouV2.avi'...
[] Cargando vídeo 45: 'PromoDeportesCuatro.avi'...
[] Cargando vídeo 46: 'Fanta.avi'...
[] Cargando vídeo 47: 'Bimbo.avi'...
[] Cargando vídeo 48: 'LineaDirecta.avi'...
[] Cargando vídeo 49: 'T5PromoLoDejoCuandoQuiera.avi'...
[] Cargando vídeo 50: 'Voltadol.avi'...
[] Cargando vídeo 51: 'CA.avi'...
[] Cargando vídeo 52: 'Europa.avi'...
[] Cargando vídeo 53: 'WallaPop.avi'...
[] Cargando vídeo 54: 'Uzu.avi'...
[] Cargando vídeo 55: 'Lancome.avi'...
[] Cargando vídeo 56: 'SolanDeCabras.avi'...
[] Cargando vídeo 57: 'Aquarius.avi'...
[] Cargando vídeo 58: 'FontVella.avi'...
[] Cargando vídeo 59: 'OperacionCamaron.avi'...
[] Cargando vídeo 60: 'Vinted.avi'...
[] Cargando vídeo 61: 'PeliculaRaya.avi'...
[] Cargando vídeo 62: 'Trombocid.avi'...
[] Cargando vídeo 63: 'T5PromoRocioLargo.avi'...
[] Cargando vídeo 64: 'LorealV2.avi'...
[] Cargando vídeo 65: 'Fairy.avi'...
[] Cargando vídeo 66: 'Vodafone.avi'...
[] Cargando vídeo 67: 'Vivus.avi'...
```

```
[] Cargando vídeo 68: 'Elvive.avi'...
[] Cargando vídeo 69: 'QualitasAuto.avi'...
[] Cargando vídeo 70: 'Nivea.avi'...
[] Cargando vídeo 71: 'IsseyMiyake.avi'...
[] Cargando vídeo 72: 'UberEats.avi'...
[] Cargando vídeo 73: 'Philadelphia.avi'...
[] Cargando vídeo 74: 'Foxy.avi'...
[] Cargando vídeo 75: 'MapfreV2.avi'...
[] Cargando vídeo 76: 'FotoCasa.avi'...
[] Cargando vídeo 77: 'Kneipp.avi'...
[] Cargando vídeo 78: 'Kellogs.avi'...
[] Cargando vídeo 79: 'DenominacionOrigen.avi'...
[] Cargando vídeo 80: 'Mapfre.avi'...
[] Cargando vídeo 81: 'LGOled.avi'...
[] Cargando vídeo 82: 'Destinia.avi'...
[] Cargando vídeo 83: 'Lowi.avi'...
[] Cargando vídeo 84: 'Ford.avi'...
[] Cargando vídeo 85: 'PorTantos.avi'...
[] Cargando vídeo 86: 'Cruzcampo.avi'...
[] Cargando vídeo 87: 'LaProvencale.avi'...
[] Cargando vídeo 88: 'RealeSeguros.avi'...
[] Cargando vídeo 89: 'FlogoProfen.avi'...
[] Cargando vídeo 90: 'AquaService.avi'...
[] Cargando vídeo 91: 'Aldi.avi'...
[] Cargando vídeo 92: 'Pelayo.avi'...
[] Cargando vídeo 93: 'PromoT5FueradeSerie.avi'...
[] Cargando vídeo 94: 'T5PromocionLoveIsInTheAir2.avi'...
[] Cargando vídeo 95: 'Vagisil.avi'...
[] Cargando vídeo 96: 'T5Transiccion.avi'...
[] Cargando vídeo 97: 'HyM.avi'...
[] Cargando vídeo 98: 'LoveIsInTheAirLargo.avi'...
[] Cargando vídeo 99: 'T5TransiccionSupervivientes.avi'...
[] Cargando vídeo 100: 'Mahou.avi'...
[] Cargando vídeo 101: 'CaixaBank.avi'...
[] Cargando vídeo 102: 'Rowenta.avi'...
[] Cargando vídeo 103: 'AquariusV2.avi'...
[] Cargando vídeo 104: 'BuscoUnChollo.avi'...
[] Cargando vídeo 105: 'Principe.avi'...
[] Cargando vídeo 106: 'Rocio.avi'...
[] Cargando vídeo 107: 'AldiV2.avi'...
[] Cargando vídeo 108: 'McDonalds.avi'...
```

```
[] Cargando vídeo 109: 'CruzRoja.avi'...
[] Cargando vídeo 110: 'GalloV2.avi'...
[] Cargando vídeo 111: 'Ambar.avi'...
[] Cargando vídeo 112: 'Opticalia.avi'...
[] Cargando vídeo 113: 'T5TransiccionLoveIsInTheAir.avi'...
[] Cargando vídeo 114: 'T5PromocionSuperLopez.avi'...
[] Cargando vídeo 115: 'NespressoV2.avi'...
[] Cargando vídeo 116: 'CruzRojaV2.avi'...
[] Cargando vídeo 117: 'AmbarV2.avi'...
[] Cargando vídeo 118: 'DonLimpio.avi'...
[] Cargando vídeo 119: 'Lidl.avi'...
[] Cargando vídeo 120: 'Carrefour.avi'...
[] Cargando vídeo 121: 'Alexa.avi'...
[] Cargando vídeo 122: 'Dodot.avi'...
[] Cargando vídeo 123: 'Decathlon.avi'...
[] Cargando vídeo 124: 'MediaMark.avi'...
[] Cargando vídeo 125: 'Gelodol.avi'...


Tamaño Base de Datos:   125
```

ANEXO B: SALIDA TRAS LA EJECUCIÓN DE SIRA-TV-2021

```
Esperando eventos...
Flujo de entrada:
        - /home/miguel/Streaming_T5/telecinco.mp4
Directorios de entrada:
        - /home/miguel/BBDD_aHash_avi/
Formatos de entrada:
        - .avi

Nuevo streaming del canal Telecinco.
--> Procesando 125 videos...

--> Analizando directorio: /home/miguel/BBDD_aHash_avi/
        [] Analizando base de datos: bbdd_pfnn.txt
        [] Analizando base de datos: bbdd_osfnn.txt

--> Videos cargados:
        [] Cargado vídeo 1:
                Nombre: 'T5TransiccionV2.avi'
                Hash:   MD5(c79ef7cb4123ef055cf4c52178914b9c)
        [] Cargado vídeo 2:
                Nombre: 'Cofidis.avi'
                Hash:   MD5(9193dd57e61458a9cafd4890cd128c87)
        [] Cargado vídeo 3:
                Nombre: 'MiCasaEsLaTuya.avi'
                Hash:   MD5(3be729b51443dfe30fc0d9c52be9911b)
        [] Cargado vídeo 4:
                Nombre: 'Meetic.avi'
                Hash:   MD5(4a5507cfc585a80946f9877e718ccfab)
        [] Cargado vídeo 5:
                Nombre: 'McDonaldsV2.avi'
                Hash:   MD5(4a4588c3d7ee78c0bd3e5136ce30d7be)
        [] Cargado vídeo 6:
                Nombre: 'PromoT5EuroV2.avi'
                Hash:   MD5(4a7f44b34b7c3d47df3095fea428eab9)
        [] Cargado vídeo 7:
                Nombre: 'VodafoneV2.avi'
                Hash:   MD5(eeb53600c87fcf44ba9813da41ce6196)
        [] Cargado vídeo 8:
                Nombre: 'Magnum.avi'
                Hash:   MD5(3955c15a330859de225d613ce6436ce8)
        [] Cargado vídeo 9:
                Nombre: 'T5PromoEuro4.avi'
                Hash:   MD5(6e725dbc97c21a69f6091d6cd2217695)
        [] Cargado vídeo 10:
                Nombre: 'T5Cine.avi'
                Hash:   MD5(d4fcba441b6e7462aab0af3e9755f570)
        [] Cargado vídeo 11:
                Nombre: 'SunBites.avi'
                Hash:   MD5(ab48f43710b50d5ecf7ff793b185d2c4)
        [] Cargado vídeo 12:
                Nombre: 'OrtizV2.avi'
                Hash:   MD5(f802f0686fd3acef1dd6f174fb28dc88)
        [] Cargado vídeo 13:
                Nombre: 'Andorra.avi'
```

```
        Hash:   MD5(a9f8677b3bc412aeff68ff56693dad14)
[] Cargado vídeo 14:
        Nombre: 'Starbucks.avi'
        Hash:   MD5(74913bf19fd9b2c1de7fb880945857bd)
[] Cargado vídeo 15:
        Nombre: 'DenominacionOrigenV2.avi'
        Hash:   MD5(b9b95adbbdda65f5c6539a5833265bd5)
[] Cargado vídeo 16:
        Nombre: 'PromoT5Euro.avi'
        Hash:   MD5(502f7a2f4b248846b62a169c385465f2)
[] Cargado vídeo 17:
        Nombre: 'Ortiz.avi'
        Hash:   MD5(1e8c4d66273e8dc68bb46b6e97070fbf)
[] Cargado vídeo 18:
        Nombre: 'Somat.avi'
        Hash:   MD5(120c739c46fe0ee3971fae2649d133f9)
[] Cargado vídeo 19:
        Nombre: 'Orange.avi'
        Hash:   MD5(f263e63004c7039da1600103e0c6b72b)
[] Cargado vídeo 20:
        Nombre: 'Bicentury.avi'
        Hash:   MD5(32daed9962d20523ca96918afecb2beb)
[] Cargado vídeo 21:
        Nombre: 'Carglass.avi'
        Hash:   MD5(3068f91763b08515ce3e6945ff134e42)
[] Cargado vídeo 22:
        Nombre: 'Beefeater.avi'
        Hash:   MD5(4143d76a99fe20cdfbefcfa868f40553)
[] Cargado vídeo 23:
        Nombre: 'Garnier.avi'
        Hash:   MD5(a4c95659d2ed1d8bcf6020bd4f1a3186)
[] Cargado vídeo 24:
        Nombre: 'T5LoveIsInTheAir4.avi'
        Hash:   MD5(a6b72d98bbbec0fb390af4dd2ab32e2d)
[] Cargado vídeo 25:
        Nombre: 'Yoigo.avi'
        Hash:   MD5(ea146ef50bf431a39154e39598506ef4)
[] Cargado vídeo 26:
        Nombre: 'MandMs.avi'
        Hash:   MD5(4c162a52260d5773fdf9fac7af404002)
[] Cargado vídeo 27:
        Nombre: 'ElPozo.avi'
        Hash:   MD5(c626f90e6463012b54781427319144e0)
[] Cargado vídeo 28:
        Nombre: 'LOreal.avi'
        Hash:   MD5(8596423bf0b16d77391f9811889d9215)
[] Cargado vídeo 29:
        Nombre: 'Puleva.avi'
        Hash:   MD5(b4d1357de3f071636298013729d70c3d)
[] Cargado vídeo 30:
        Nombre: 'ColchonesElCorteIngles.avi'
        Hash:   MD5(5af920303aa6377bc96f5f44a9fbc8db)
[] Cargado vídeo 31:
        Nombre: 'LoveIsInTheAir.avi'
        Hash:   MD5(69183cf6796123f6b3ce92b47d37a69b)
[] Cargado vídeo 32:
        Nombre: 'Gallo.avi'
        Hash:   MD5(0c74ed1b2348437d3be2eb69d6f22fed)
[] Cargado vídeo 33:
        Nombre: 'PrecioJusto.avi'
        Hash:   MD5(0ecb1cc7b70395e4488353d427295a34)
[] Cargado vídeo 34:
        Nombre: 'Norit.avi'
```

```
        Hash:   MD5(8909e015d44e3ee1920fcd396641405a)
[] Cargado vídeo 35:
        Nombre: 'DonSimon.avi'
        Hash:   MD5(4ee3a4fa8c5f9d10580b7798892efb18)
[] Cargado vídeo 36:
        Nombre: 'LowiV2.avi'
        Hash:   MD5(7f07c6d8d27c6fe75aea919400f7fea0)
[] Cargado vídeo 37:
        Nombre: 'Alhambra.avi'
        Hash:   MD5(33727e88f111e1b9b39f9f30e000b412)
[] Cargado vídeo 38:
        Nombre: 'OrangeLargo.avi'
        Hash:   MD5(9c3aa01b03c8fcafebe6d464aea4df69)
[] Cargado vídeo 39:
        Nombre: 'T5PromoEuroV3.avi'
        Hash:   MD5(0347d1d1a6ff46144a6b97abb8908b41)
[] Cargado vídeo 40:
        Nombre: 'T5PromocionRocioLargo.avi'
        Hash:   MD5(567d6783d85dcd7b6c39b508b5b4d4bc)
[] Cargado vídeo 41:
        Nombre: 'T5Rocio3.avi'
        Hash:   MD5(8db5f2e1b447994decd8527ae7b944f4)
[] Cargado vídeo 42:
        Nombre: 'Nespresso.avi'
        Hash:   MD5(08a58b9c56e712b3925e1fc433b8e819)
[] Cargado vídeo 43:
        Nombre: 'T5PromoSupervivientes.avi'
        Hash:   MD5(c8aeb1fe9b247cab48c59910f6bbef41)
[] Cargado vídeo 44:
        Nombre: 'MahouV2.avi'
        Hash:   MD5(b2a86b1911e64b4470e17503d4fe9c6a)
[] Cargado vídeo 45:
        Nombre: 'PromoDeportesCuatro.avi'
        Hash:   MD5(ecc32b31d2515ab6e9c5294034b71076)
[] Cargado vídeo 46:
        Nombre: 'Fanta.avi'
        Hash:   MD5(94a2f2e86c6436f41750517af9b36aee)
[] Cargado vídeo 47:
        Nombre: 'Bimbo.avi'
        Hash:   MD5(9a100ff08255babd729886fca6e42e5d)
[] Cargado vídeo 48:
        Nombre: 'LineaDirecta.avi'
        Hash:   MD5(603337d024832f4b9b04739394297235)
[] Cargado vídeo 49:
        Nombre: 'T5PromoLoDejoCuandoQuiera.avi'
        Hash:   MD5(a2dd922aef2a6a2462d0c58a03507f4c)
[] Cargado vídeo 50:
        Nombre: 'Voltadol.avi'
        Hash:   MD5(97f7d5c48fb2ad095f67d14c4dbe8647)
[] Cargado vídeo 51:
        Nombre: 'CA.avi'
        Hash:   MD5(19f89d957536f1fde7c950f667dbd0d2)
[] Cargado vídeo 52:
        Nombre: 'Europa.avi'
        Hash:   MD5(f1b6bb39893853c5eda688de587fe15d)
[] Cargado vídeo 53:
        Nombre: 'WallaPop.avi'
        Hash:   MD5(8626fc79ad38b71f5bc88d4dce6a220a)
[] Cargado vídeo 54:
        Nombre: 'Uzu.avi'
        Hash:   MD5(453680aa690700fbf92996a1e1218e25)
[] Cargado vídeo 55:
        Nombre: 'Lancome.avi'
```

```
        Hash:   MD5(e5f1d38caf604cacb64d15c602a41619)
[] Cargado vídeo 56:
        Nombre: 'SolanDeCabras.avi'
        Hash:   MD5(706ec80290b7bdd13763f02cc4e12df8)
[] Cargado vídeo 57:
        Nombre: 'Aquarius.avi'
        Hash:   MD5(949d9d2eb6687670ac4ff732a16eee18)
[] Cargado vídeo 58:
        Nombre: 'FontVella.avi'
        Hash:   MD5(c354fa3890b3e9119a6d2ebe55570016)
[] Cargado vídeo 59:
        Nombre: 'OperacionCamaron.avi'
        Hash:   MD5(4ed679d01e69ba6e42cd2fc82ccaf0d8)
[] Cargado vídeo 60:
        Nombre: 'Vinted.avi'
        Hash:   MD5(ea37097ccb73e65f815ef36e9261a9ae)
[] Cargado vídeo 61:
        Nombre: 'PeliculaRaya.avi'
        Hash:   MD5(1cc644310a4524d0b329e49dde11f837)
[] Cargado vídeo 62:
        Nombre: 'Trombocid.avi'
        Hash:   MD5(37c6108ac9f9447d0b8cbe0cd4318a87)
[] Cargado vídeo 63:
        Nombre: 'T5PromoRocioLargo.avi'
        Hash:   MD5(d4c5ec713a8f88e6e1c69bd73fffa5ae)
[] Cargado vídeo 64:
        Nombre: 'LorealV2.avi'
        Hash:   MD5(40bad3878bad043d56468b005444ff20)
[] Cargado vídeo 65:
        Nombre: 'Fairy.avi'
        Hash:   MD5(38ba76fd43649e68a57a9a9c298536ed)
[] Cargado vídeo 66:
        Nombre: 'Vodafone.avi'
        Hash:   MD5(d5e46504387e91e5af6fd1ab5ba35b88)
[] Cargado vídeo 67:
        Nombre: 'Vivus.avi'
        Hash:   MD5(4b769e9fe0a980ef961fd276535804b7)
[] Cargado vídeo 68:
        Nombre: 'Elvive.avi'
        Hash:   MD5(127545054ca34785487273ed077620e8)
[] Cargado vídeo 69:
        Nombre: 'QualitasAuto.avi'
        Hash:   MD5(fd7682ea43d419873470e4877de02550)
[] Cargado vídeo 70:
        Nombre: 'Nivea.avi'
        Hash:   MD5(6378cd5f45f47c535f9241b900c4eb31)
[] Cargado vídeo 71:
        Nombre: 'IsseyMiyake.avi'
        Hash:   MD5(1778121d7a8163fe6f71e2da538f344b)
[] Cargado vídeo 72:
        Nombre: 'UberEats.avi'
        Hash:   MD5(2d7ca91507548d3308b0e97c00a64502)
[] Cargado vídeo 73:
        Nombre: 'Philadelphia.avi'
        Hash:   MD5(c79e7b9c8babcf532545c0f8ce7b39d7)
[] Cargado vídeo 74:
        Nombre: 'Foxy.avi'
        Hash:   MD5(ab5c8fe7b0aab92f2735d9c7a3de2c68)
[] Cargado vídeo 75:
        Nombre: 'MapfreV2.avi'
        Hash:   MD5(802a69597b93d4a10000371d0a00a896)
[] Cargado vídeo 76:
        Nombre: 'FotoCasa.avi'
```

```
        Hash:   MD5(6b0df0f9bf240a455f8bca923c71cbbf)
[] Cargado vídeo 77:
        Nombre: 'Kneipp.avi'
        Hash:   MD5(6574b5d4a86021db4f024fc5f0e37cdf)
[] Cargado vídeo 78:
        Nombre: 'Kellogs.avi'
        Hash:   MD5(c01bbb1f419053d48387d514f7654944)
[] Cargado vídeo 79:
        Nombre: 'DenominacionOrigen.avi'
        Hash:   MD5(46dafe8ccf81603dcf94d1a538995456)
[] Cargado vídeo 80:
        Nombre: 'Mapfre.avi'
        Hash:   MD5(8991e35baabc5b042eea1e810e044eea)
[] Cargado vídeo 81:
        Nombre: 'LGOled.avi'
        Hash:   MD5(72964ebbd5c8c7d44d42023bf9ff0817)
[] Cargado vídeo 82:
        Nombre: 'Destinia.avi'
        Hash:   MD5(fb552501d0f2a2325824e87ca1b8ca37)
[] Cargado vídeo 83:
        Nombre: 'Lowi.avi'
        Hash:   MD5(78896de82f620f7735f21cb534bb5dde)
[] Cargado vídeo 84:
        Nombre: 'Ford.avi'
        Hash:   MD5(b0211d2ae530daac732c1b7b3b8e81b0)
[] Cargado vídeo 85:
        Nombre: 'PorTantos.avi'
        Hash:   MD5(bb25132facbc4e39138f9d925748ae49)
[] Cargado vídeo 86:
        Nombre: 'Cruzcampo.avi'
        Hash:   MD5(bb356680ba25b8e24245543fdffcffd9)
[] Cargado vídeo 87:
        Nombre: 'LaProvencale.avi'
        Hash:   MD5(m4712204db900fbd3fe0d3aff622640ec)
[] Cargado vídeo 88:
        Nombre: 'RealeSeguros.avi'
        Hash:   MD5(0cafa34e0a33db91da82c43f9d72a18d)
[] Cargado vídeo 89:
        Nombre: 'FlogoProfen.avi'
        Hash:   MD5(dbea552f42d533848ed4950e6fdb9ba0)
[] Cargado vídeo 90:
        Nombre: 'AquaService.avi'
        Hash:   MD5(0c239b42dc2483d474f8ed41e47de9ad)
[] Cargado vídeo 91:
        Nombre: 'Aldi.avi'
        Hash:   MD5(035422e96fb852f01631234748cba702)
[] Cargado vídeo 92:
        Nombre: 'Pelayo.avi'
        Hash:   MD5(ba6b4fa9810aa6c0293ad9ca7da87143)
[] Cargado vídeo 93:
        Nombre: 'PromoT5FueradeSerie.avi'
        Hash:   MD5(56b31ba65351f24be8b08c68b085b4da)
[] Cargado vídeo 94:
        Nombre: 'T5PromocionLoveIsInTheAir2.avi'
        Hash:   MD5(795f2408593ddd76b82fc897a236d80a)
[] Cargado vídeo 95:
        Nombre: 'Vagisil.avi'
        Hash:   MD5(9f54306ad4f5dfe2351a5e3683e313fe)
[] Cargado vídeo 96:
        Nombre: 'T5Transiccion.avi'
        Hash:   MD5(06a68cfafb7b92401cacb2c9777b8a22)
[] Cargado vídeo 97:
        Nombre: 'HyM.avi'
```

```
        Hash:   MD5(a7515a66838f433f4a1a701affdab898)
[] Cargado vídeo 98:
        Nombre: 'LoveIsInTheAirLargo.avi'
        Hash:   MD5(719a1872616378a489687334be15dbc8)
[] Cargado vídeo 99:
        Nombre: 'T5TransiccionSupervivientes.avi'
        Hash:   MD5(85cb374ca1c906ef1a487b4dd207b82b)
[] Cargado vídeo 100:
        Nombre: 'Mahou.avi'
        Hash:   MD5(6e906c85fc7a129abeb375538c8ef26b)
[] Cargado vídeo 101:
        Nombre: 'CaixaBank.avi'
        Hash:   MD5(1c45f4cc0cd2fb4a9ed71272d47eb965)
[] Cargado vídeo 102:
        Nombre: 'Rowenta.avi'
        Hash:   MD5(2910bdade742febf32c7bcc57213b7d2)
[] Cargado vídeo 103:
        Nombre: 'AquariusV2.avi'
        Hash:   MD5(e1e8d6820580f5b1ee8e322d06d9e4bf)
[] Cargado vídeo 104:
        Nombre: 'BuscoUnChollo.avi'
        Hash:   MD5(9b4286026f7b3502b80cbb6251fd8c40)
[] Cargado vídeo 105:
        Nombre: 'Principe.avi'
        Hash:   MD5(21d83b61288582c33689ed3ffade752b)
[] Cargado vídeo 106:
        Nombre: 'Rocio.avi'
        Hash:   MD5(20ce39d94aee49eb95d4b927ab04103b)
[] Cargado vídeo 107:
        Nombre: 'AldiV2.avi'
        Hash:   MD5(f7fdb0ab8f7de92dadff8c5c192ac09b)
[] Cargado vídeo 108:
        Nombre: 'McDonalds.avi'
        Hash:   MD5(4526f820fe3d2bfcefd9da412c695277)
[] Cargado vídeo 109:
        Nombre: 'CruzRoja.avi'
        Hash:   MD5(6813b31af3e8e67d6290f90443936703)
[] Cargado vídeo 110:
        Nombre: 'GalloV2.avi'
        Hash:   MD5(687a58dd8d85cc2a9bc710c5b3f7de31)
[] Cargado vídeo 111:
        Nombre: 'Ambar.avi'
        Hash:   MD5(227e63c56606f1cd8a4f28f7044404e0)
[] Cargado vídeo 112:
        Nombre: 'Opticalia.avi'
        Hash:   MD5(6b8a53dda326cb2885cad8adb18fc96f)
[] Cargado vídeo 113:
        Nombre: 'T5TransiccionLoveIsInTheAir.avi'
        Hash:   MD5(42b93560ab4569f881c5b8a4c4eca036)
[] Cargado vídeo 114:
        Nombre: 'T5PromocionSuperLopez.avi'
        Hash:   MD5(020e55bb223b81bd869e829feffb700e)
[] Cargado vídeo 115:
        Nombre: 'NespressoV2.avi'
        Hash:   MD5(feb4fca63c5e1618622c77e08eb682b2)
[] Cargado vídeo 116:
        Nombre: 'CruzRojaV2.avi'
        Hash:   MD5(ec4418e21484136270219baf8f0fc5df)
[] Cargado vídeo 117:
        Nombre: 'AmbarV2.avi'
        Hash:   MD5(3cb89ba70d84c31f168bd4d9a4999a4c)
[] Cargado vídeo 118:
        Nombre: 'DonLimpio.avi'
```

```
                Hash:   MD5(e3d7a816f6f5ee4a81cc64fcbf745303)
        [] Cargado vídeo 119:
                Nombre: 'Lidl.avi'
                Hash:   MD5(341b576326532d70636e92e17040c577)
        [] Cargado vídeo 120:
                Nombre: 'Carrefour.avi'
                Hash:   MD5(a943621eb72029190b70fbc7d8f37447)
        [] Cargado vídeo 121:
                Nombre: 'Alexa.avi'
                Hash:   MD5(ac2365eff371bd73c6081e6a9dbe80b1)
        [] Cargado vídeo 122:
                Nombre: 'Dodot.avi'
                Hash:   MD5(6bb9797a1ba6805bd1b82bdd87676ace)
        [] Cargado vídeo 123:
                Nombre: 'Decathlon.avi'
                Hash:   MD5(2a9d1d9b33859c4b5b27cdbb32f0a45f)
        [] Cargado vídeo 124:
                Nombre: 'MediaMark.avi'
                Hash:   MD5(e0a7c656ab7251ca7f4cf43f70a1d870)
        [] Cargado vídeo 125:
                Nombre: 'Gelodol.avi'
                Hash:   MD5(4f080b09e33c374f98c65abf4f204676)

Tamaño Base de Datos: 125

--> Configurando parámertos
        [ ] Capacidad de los buffers de Video: 750 frames
        [ ] Capacidad de los buffers de Audio: 1406 frames
        [ ] Framerate: 25 fps
        [ ] Spot Length: 30 segundos
        [ ] Guardando frames: No

Leyendo frames hasta el fin de retransmisión...

--> Posible anuncio tras 26584 frames.
--> Asignado buffer Nº 1/20
--> Buscando coindidencias con la BBDD...
[comparadorHuella @ Buffer 01] [/] Obteniendo mejor candidato del frame PFNN
2...
[comparadorHuella @ Buffer 01] [/] Candidato 01/01 (#001): Distancia de
hamming: 0
[compruebaVideoID @ Buffer 01] [+] El anuncio corresponde al anuncio 1 de la
BBDD: T5TransiccionV2.avi - c79ef7cb4123ef055cf4c52178914b9c
[compruebaVideoID @ Buffer 01] [/] Liberando buffer

--> Posible anuncio tras 26696 frames.
--> Asignado buffer Nº 1/20
--> Buscando coindidencias con la BBDD...
[comparadorHuella @ Buffer 01] [/] Obteniendo mejor candidato del frame PFNN
12...
[comparadorHuella @ Buffer 01] [/] Candidato 01/01 (#071): Distancia de
hamming: 2
[compruebaVideoID @ Buffer 01] [+] El anuncio corresponde al anuncio 71 de la
BBDD: IsseyMiyake.avi - 1778121d7a8163fe6f71e2da538f344b
[compruebaVideoID @ Buffer 01] [/] Liberando buffer

--> Posible anuncio tras 27455 frames.
--> Asignado buffer Nº 1/20
--> Buscando coindidencias con la BBDD...
[comparadorHuella @ Buffer 01] [/] Obteniendo mejor candidato del frame PFNN
9...
[comparadorHuella @ Buffer 01] [/] Candidato 01/01 (#096): Distancia de
hamming: 0
```

```
[compruebaVideoID @ Buffer 01] [+] El anuncio corresponde al anuncio 96 de la
BBDD: T5Transiccion.avi - 06a68cfafb7b92401cacb2c9777b8a22
[compruebaVideoID @ Buffer 01] [/] Liberando buffer

--> Posible anuncio tras 27574 frames.
--> Asignado buffer N° 1/20
--> Buscando coindidencias con la BBDD...
[comparadorHuella @ Buffer 01] [/] Obteniendo mejor candidato del frame PFNN
19...
[comparadorHuella @ Buffer 01] [/] Obteniendo mejor candidato del frame OSFNN
44...
[compruebaVideoID @ Buffer 01] [-] Anuncio no encontrado
[compruebaVideoID @ Buffer 01] [/] Liberando buffer

--> Posible anuncio tras 27912 frames.
--> Asignado buffer N° 2/20
--> Buscando coindidencias con la BBDD...
[comparadorHuella @ Buffer 02] [/] Obteniendo mejor candidato del frame PFNN
3...
[comparadorHuella @ Buffer 02] [/] Candidato 01/01 (#031): Distancia de
hamming: 0
[compruebaVideoID @ Buffer 02] [+] El anuncio corresponde al anuncio 31 de la
BBDD: LoveIsInTheAir.avi - 69183cf6796123f6b3ce92b47d37a69b
[compruebaVideoID @ Buffer 02] [/] Liberando buffer

--> Posible anuncio tras 28023 frames.
--> Asignado buffer N° 1/20
--> Buscando coindidencias con la BBDD...
[comparadorHuella @ Buffer 01] [/] Obteniendo mejor candidato del frame PFNN
11...
[comparadorHuella @ Buffer 01] [/] Candidato 01/01 (#075): Distancia de
hamming: 1
[compruebaVideoID @ Buffer 01] [+] El anuncio corresponde al anuncio 75 de la
BBDD: MapfreV2.avi - 802a69597b93d4a10000371d0a00a896
[compruebaVideoID @ Buffer 01] [/] Liberando buffer

--> Posible anuncio tras 28534 frames.
--> Asignado buffer N° 1/20
--> Buscando coindidencias con la BBDD...
[comparadorHuella @ Buffer 01] [/] Obteniendo mejor candidato del frame PFNN
11...
[comparadorHuella @ Buffer 01] [/] Candidato 01/01 (#101): Distancia de
hamming: 1
[compruebaVideoID @ Buffer 01] [+] El anuncio corresponde al anuncio 101 de
la BBDD: CaixaBank.avi - 1c45f4cc0cd2fb4a9ed71272d47eb965
[compruebaVideoID @ Buffer 01] [/] Liberando buffer


// . . . . . . . . . . . //


--> Posible anuncio tras 277591 frames.
--> Asignado buffer N° 1/20
--> Buscando coindidencias con la BBDD...
[comparadorHuella @ Buffer 01] [/] Obteniendo mejor candidato del frame PFNN
10...
[comparadorHuella @ Buffer 01] [/] Candidato 01/01 (#009): Distancia de
hamming: 0
[compruebaVideoID @ Buffer 01] [+] El anuncio corresponde al anuncio 9 de la
BBDD: T5PromoEuro4.avi - 6e725dbc97c21a69f6091d6cd2217695
[compruebaVideoID @ Buffer 01] [/] Liberando buffer

--> Posible anuncio tras 277701 frames.
```

```
--> Asignado buffer Nº 1/20
--> Buscando coindidencias con la BBDD...
[comparadorHuella @ Buffer 01] [/] Obteniendo mejor candidato del frame PFNN
10...
[comparadorHuella @ Buffer 01] [/] Obteniendo mejor candidato del frame OSFNN
35...
[compruebaVideoID @ Buffer 01] [-] Anuncio no encontrado
[compruebaVideoID @ Buffer 01] [/] Liberando buffer
Streaming del canal Telecinco finalizado.

--> Fin de la lectura.


[ ] Anuncios totales: 165
[+] Anuncios encontrados: 159 (96.363636%)
[-] Anuncios desconocidos: 6 (3.636363%)
[ ] Segmentos de programación: 9
```

ANEXO C: TABLAS DE RESULTADOS TRAS LA EJECUCIÓN DE LA APLICACIÓN SOBRE CADA STREAMING

Pruebas sobre el streaming original:

Método de hashing que implementa la DCT:

Tabla C-1-a. Resultados obtenidos tras la ejecución del streaming sin distorsión - DCT

Umbral Hamming (%)	Verdaderos Positivos (**VP**)	Falsos Positivos (**FP**)	Verdaderos Negativos (**VN**)	Falsos Negativos (**FN**)
1	58	0	15	101
3	145	0	15	14
5	154	0	15	5
10	158	0	15	1
18	159	0	15	0

Método ahash

Tabla C-1-b. Resultados obtenidos tras la ejecución del streaming sin distorsión - ahash

Umbral Hamming (%)	Verdaderos Positivos (**VP**)	Falsos Positivos (**FP**)	Verdaderos Negativos (**VN**)	Falsos Negativos (**FN**)
1	157	0	15	2
3	159	0	15	0
5	159	0	15	0
10	159	8	7	0
18	159	8	7	0

Pruebas sobre el streaming con disminución en la tasa de fotogramas por segundo (15fps).

Método de hashing que implementa la DCT:

Tabla C-2-a. Resultados obtenidos tras la ejecución del streaming con reducción de FPS - DCT

Umbral Hamming (%)	Verdaderos Positivos (**VP**)	Falsos Positivos (**FP**)	Verdaderos Negativos (**VN**)	Falsos Negativos (**FN**)
1	4	0	15	155
3	67	1	15	91
5	95	1	15	63
10	124	1	15	35
18	141	11	10	12

Método ahash

Tabla C-2-b. Resultados obtenidos tras la ejecución del streaming con reducción de FPS - ahash

Umbral Hamming (%)	Verdaderos Positivos (**VP**)	Falsos Positivos (**FP**)	Verdaderos Negativos (**VN**)	Falsos Negativos (**FN**)
1	103	0	15	56
3	134	0	15	25
5	135	0	15	24
10	148	1	14	11
18	153	11	10	0

Pruebas sobre el streaming al que se le ha añadido Rudio Blanco Gaussiano.

Método de hashing que implementa la DCT:

Tabla C-3-a. Resultados obtenidos tras la ejecución del streaming con Ruido Gaussiano Blanco - DCT

Umbral Hamming (%)	Verdaderos Positivos **(VP)**	Falsos Positivos **(FP)**	Verdaderos Negativos **(VN)**	Falsos Negativos **(FN)**
1	0	0	15	159
3	69	0	15	90
5	133	0	15	26
10	158	0	15	11
18	159	0	15	0

Método ahash

Tabla C-3-b. Resultados obtenidos tras la ejecución del streaming con Ruido Gaussiano Blanco - ahash

Umbral Hamming (%)	Verdaderos Positivos **(VP)**	Falsos Positivos **(FP)**	Verdaderos Negativos **(VN)**	Falsos Negativos **(FN)**
1	128	0	15	31
3	157	0	15	2
5	158	2	13	1
10	159	3	12	0
18	159	3	12	0

Pruebas sobre el streaming al que se le ha añadido el filtro de difuminado Gaussian blur.

Método de hashing que implementa la DCT:

Tabla C-4-a. Resultados obtenidos tras la ejecución del streaming con el filtro Gaussian blur- DCT

Umbral Hamming (%)	Verdaderos Positivos (**VP**)	Falsos Positivos (**FP**)	Verdaderos Negativos (**VN**)	Falsos Negativos (**FN**)
1	1	0	15	158
3	65	0	15	94
5	132	0	15	27
10	151	0	15	8
18	159	0	15	0

Método ahash:

Tabla C-4-b. Resultados obtenidos tras la ejecución del streaming con el filtro Gaussian blur- ahash

Umbral Hamming (%)	Verdaderos Positivos (**VP**)	Falsos Positivos (**FP**)	Verdaderos Negativos (**VN**)	Falsos Negativos (**FN**)
1	78	0	15	81
3	154	0	15	5
5	158	0	15	1
10	159	1	14	0
18	159	9	6	0

Pruebas sobre el streaming después de realizarle un aumento en la corrección gamma (+30%).

Método de hashing que implementa la DCT:

Tabla C-5-a. Resultados obtenidos tras la ejecución del streaming con aumento en la corrección gamma- DCT

Umbral Hamming (%)	Verdaderos Positivos (**VP**)	Falsos Positivos (**FP**)	Verdaderos Negativos (**VN**)	Falsos Negativos (**FN**)
1	0	0	15	159
3	45	0	15	114
5	135	0	15	24
10	156	0	15	3
18	156	3	15	0

Método ahash:

Tabla C-5-b. Resultados obtenidos tras la ejecución del streaming con aumento en la corrección gamma - ahash

Umbral Hamming (%)	Verdaderos Positivos (**VP**)	Falsos Positivos (**FP**)	Verdaderos Negativos (**VN**)	Falsos Negativos (**FN**)
1	0	0	0	0
3	144	0	15	15
5	159	0	15	0
10	159	2	13	0
18	159	4	11	0