ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE
TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Trabajo Fin de Grado en Telemática

HERRAMIENTA EN MATLAB PARA SEGUIMIENTO DE PATRONES EN SECUENCIAS DE VÍDEO

AUTOR: JOSÉ BARTOLOMÉ ASENSIO GARCÍA
DIRECTOR: JUAN CARLOS TRILLO MOYA
Noviembre / 2012

Herramienta en Matlab para seguimiento de patrones en secuencias de vídeo

Autor: José Bartolomé Asensio García
Director: Juan Carlos Trillo Moya

Noviembre 2012

Índice de Tablas

Índice de figuras

Índice general

Capítulo 1

Introducción

El objetivo de este proyecto es construir un reconocedor de patrones de vídeo, cuyas aplicaciones se extienden desde el campo de la enseñanza de las matemáticas y la informática hasta la videovigilancia. Su implementación está basada en la desigualdad matemática de Cauchy-Schwarz que veremos más adelante.

Vamos a implementar el algoritmo resultante en código MATLAB, ya que es una herramienta muy útil para el desarrollo ágil de los programas.

Esta herramienta nace para complementar los casos uno y dos dimensional, ya que en un anterior estudio analizamos las transformaciones de la imagen en el caso bidimensional y de audio en el caso unidimensional.

Actualmente existen diversidad de aplicaciones informáticas, en diversos campos como medicina o sistemas militares, que trabajan con secuencias de vídeo y necesitan un procesamiento de las mismas en tiempo real. En este estudio, nos vamos a centrar en el seguimiento y localización de un patrón, seleccionado previamente, para posteriormente estudiar la fiabilidad del algoritmo propuesto, analizando en particular el caso de la existencia de oclusiones de dicho patrón en el vídeo.

En el caso tridimensional extraemos los distintos fotogramas del vídeo ya sea a color o en tonalidades de grises para realizar nuestros experimentos. Nuestro objetivo es encontrar un patrón (template) dentro de cada imagen original. Para ello utilizaremos el filtro basado en la citada desigualdad de Cauchy-Schwarz, complementado con otros criterios de búsqueda como la

cercanía espacial o de la media estadística de los píxeles vecinos.

Una de nuestras finalidades es testar la robusted del algoritmo frente al movimiento de la imagen original del vídeo, en la cual, con la sensación de movimiento se irá describiendo una cierta deformación del patrón. Analizaremos los resultados de la búsqueda y realizaremos un estudio de hasta qué punto el algoritmo testado es ciertamente fiable.

En la Figura 1.1 vemos una imagen extraída del partido de la final del mundial de fútbol 2010, donde usamos el estudio el balón como patrón para el estudio. Nuestro patrón, que podemos ver en la Figura 1.2, estará encuadrado en rojo. El algoritmo va a buscar por nosotros la localización del template durante toda la jugada, tal y como se muestra en la Figura 1.3 hasta el momento en el que por diversos motivos se produzca una oclusión, que en este caso, será el golpeo del balón por el jugador, lo que nos hará desaparecer el template de la imagen.

Los conceptos de digitalizar imágenes en escáneres y convertir señales de vídeo a digital anteceden al concepto de tomar cuadros fijos digitalizando así señales de una matriz de elementos sensores discretos. Eugene F. Lally del Jet Propulsion Laboratory publicó la primera descripción de como producir fotos fijas en un dominio digital usando un fotosensor en mosaico. El propósito era proporcionar información de navegación a los astronautas a bordo durante misiones espaciales. La matriz en mosaico registraba periódicamente fotos fijas de las localizaciones de estrellas y planetas durante el tránsito y cuando se acercaba a un planeta, proporcionaba información adicional de distancias para el orbitaje y servía como guía para el aterrizaje. El concepto incluyó elementos de diseño que presagiaban la primera cámara fotográfica digital.

El ingeniero ruso Vladimir Kozmich Zvorykin desarrolló en 1923 un sistema de captación de imágenes que tres años después fue perfeccionado por el ingeniero escocés John Logie Baird quien hizo las primeras demostraciones de transmitir imágenes de 3'8x5 cm. a una definición de 30 líneas.

En la época de los 80 del siglo XX, se desarrollaron transductores de estado sólido: los CCDs (Dispositivos de cargas acopladas). Ellos sustituyeron muy ventajosamente a los tubos electrónicos, propiciando una disminución en el tamaño y el peso de las cámaras de vídeo. Además proporcionaron una mayor calidad y fiabilidad, aunque con una exigencia más elevada en

las prestaciones de las ópticas utilizadas.

La televisión en blanco y negro, que utiliza únicamente la información de la luz de una imagen, la luminancia, utiliza cámaras de un solo canal de captación. Los sistemas para televisión en color, que necesitan captar las características que diferencian los colores, la crominancia, usan tres canales; cada uno de ellos destinado a la captura de cada color primario.

De especial significación han sido también los numerosos trabajos que han usado técnicas de representación del conocimiento para los problemas de interpretación de imágenes, en relación con aplicaciones de ambiente industrial, iniciando una fuerte hibridación entre las técnicas de la Inteligencia Artificial para la representación del conocimiento y los técnicas de interpretación de escenas a partir de imágenes digitales.

Hasta el siglo XXI, las soluciones disponibles en aquel momento (como los paquetes de software para la creación de presentaciones de diapositivas sencillas o los productos Flash en línea), no permitían realmente realizar tareas de montaje, ni sincronizar la música de fondo con la secuencia de imágenes. Por otro lado, los paquetes de software profesionales utilizados por los diseñadores gráficos y de animación en 3D y por los productores de vídeos resultaban caros y exigían un alto nivel de conocimientos y mucho tiempo de dedicación por parte del usuario.

Vemos pues el creciente auge de las técnicas de procesamiento digital de imágenes y su notable importancia en el mundo que nos rodea. A lo largo de este proyecto nosotros vamos a desarrollar un estudio sobre un campo concreto como es el del reconocimiento de patrones en técnicas de vídeos apoyándonos de los anteriores resultados extraídos del estudio con variables unidimensional y dosdimensional respectivamente, siempre abordando el desarrollo asociado a la desigualdad matemática de Cauchy-Schwarz (ver [3]).

Por completitud del trabajo presentado y con la intención de hacerlo más autocontenido incluímos también las partes del caso unidimensional y dos dimensional que consideramos importantes para el desarrollo del caso tridimensional.

Los experimentos numéricos los realizamos en el lenguaje de programación Matlab (ver [7], [10]), ya que este lenguaje de alto nivel posee un entorno interactivo para el cálculo numérico, la visualización y la programa-

Figura 1.1: Momento del pase de Cesc

Figura 1.2: Template en el que se puede ver el balón de fútbol

ción. Además, sus herramientas y las funciones matemáticas incorporadas permiten explorar diversos enfoques y se puede utilizar en una gran variedad de aplicaciones, tales como procesamiento de señales y comunicaciones, procesamiento de imagen y vídeo, sistemas de control, pruebas y medidas, finanzas computacionales y biología computacional. Más de un millón de ingenieros y científicos de la industria y la educación lo utilizan.

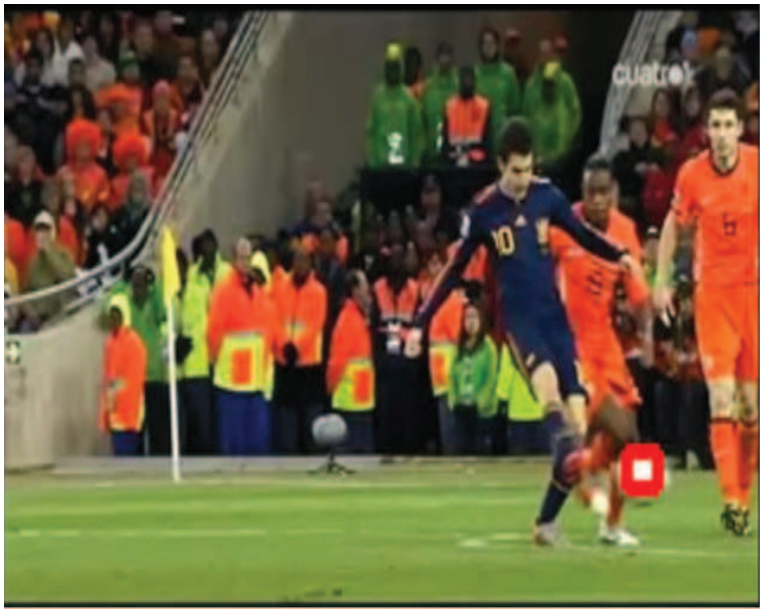


Figura 1.3: Encuentra el patrón en el momento del pase

En el capítulo 2 se recuerdan las nociones de producto escalar, espacio prehilbertiano y espacio de Hilbert, así como la famosa desigualdad de Cauchy-Schwarz, que será la motivación para la definición del filtro que permite llevar a cabo el reconocimiento de patrones. Finalizaremos el capítulo con la definición del algoritmo 1D a que da lugar dicho filtro.

En el capítulo 3 se utiliza la definición del algoritmo en $1D$ para aplicarlo al procesamiento de las señales de audio. Explicaremos el programa de audio, en el cual introducimos una señal (puede ser cualquier pista de música por ejemplo), que sometemos a diversos cambios aplicando ruido, y un template (muestra de la señal anterior) dando como resultado la localización del patrón dentro de la pista de audio.

En el capítulo 4 se define el algoritmo en $2D$ para aplicarlo al procesamiento de imágenes digitales. También en este caso explicaremos el programa realizado de reconocimiento de patrones. En este programa se introduce una imagen (bien a color, bien en tonos de grises) y un template (patrón o imagen de búsqueda), obteniendo como resultado la localización del template dentro de la imagen.

En el capítulo 5 analizaremos diversos tipos de transformación sobre imágenes digitales que son de uso muy usual. Explicaremos cada transformación, de manera teórica y con casos prácticos. Entre las transformaciones consideradas están la rotación, los filtros de suavidad como el de mediana y el de vecindad, la adición de ruido bien sea gaussiano, multiplicativo o sal y pimienta, la compresión de imágenes digitales en formato JPEG, y para terminar nos centraremos en la ecualización de histogramas. La imagen original puede someterse a diversos cambios aplicando estas transformaciones antes de realizar la búsqueda del patrón. Estudiaremos la robustez del algoritmo de reconocimiento de patrones frente a estas transformaciones.

En el capítulo 6 explicaremos en detalle cómo llevamos a cabo el seguimiento de un patrón en una secuencia de vídeo, ejemplificando varios casos prácticos.

En el capítulo 7 nos centraremos en el desarrollo de los programas en Matlab, y escribiremos el código fuente de los programas para los distintos algoritmos que hemos realizado. Los programas se facilitarán tanto con interfaz gráfica como sin ella.

Para terminar expondremos unas breves conclusiones del trabajo en el capítulo 8.

Capítulo 2

Reconocimiento de Patrones

El reconocimiento de patrones, también llamado lectura de patrones, identificación de figuras y reconocimiento de formas consiste en la búsqueda de una muestra o patrón dentro de una se nal de mayor tama no. No sólo es un campo de la informática sino un proceso fundamental que se encuentra en casi todas las acciones humanas.

El punto esencial del reconocimiento de patrones es la clasificación: se quiere clasificar una señal dependiendo de sus características. Señales, características y clases pueden ser de cualquiera forma, por ejemplo se puede clasificar imágenes digitales de letras en las clases «A» a «Z» dependiente de sus píxeles o se puede clasificar ruidos de cantos de los pájaros en clases de órdenes aviares dependiente de las frecuencias.

El objetivo es clasificar patrones con base en un conocimiento a priori o información estadística extraída de los patrones. Los patrones a clasificar suelen ser grupos de medidas u observaciones, definiendo puntos en un espacio multidimensional apropiado.

Un sistema de reconocimiento de patrones completo consiste en un sensor que recoge las observaciones a clasificar, un sistema de extracción de características que transforma la información observada en valores numéricos o simbólicos, y un sistema de clasificación o descripción que, basado en las características extraídas, clasifica la medición.

La clasificación utiliza habitualmente uno de las siguientes procedimientos: clasificación estadística (o teoría de la decisión) o clasificación sintáctica

(o estructural). El reconocimiento estadístico de patrones está basado en las características estadísticas de los patrones, asumiendo que han sido generados por un sistema probabilístico. El reconocimiento estructural de patrones está basado en las relaciones estructurales de las características.

Para la clasificación se puede usar un conjunto de aprendizaje, del cual ya se conoce la clasificación de la información a priori y se usa para entrenar al sistema, siendo la estrategia resultante conocida como aprendizaje supervisado. El aprendizaje puede ser también no supervisado, el sistema no tiene un conjunto para aprender a clasificar la información a priori, sino que se basa en cálculos estadísticos para clasificar los patrones.

Entre las aplicaciones del reconocimiento de patrones son el reconocimiento de voz, la clasificación de documentos (por ejemplo spam/no spam), el reconocimiento de escritura, reconocimiento de caras humanas y muchas más. Los dos últimos ejemplos son representativos del análisis de imágenes, un subconjunto del reconocimiento de patrones que toma imágenes digitales como entradas del sistema.

El reconocimiento de patrones es más complejo cuando se usan plantillas para generar variantes. Por ejemplo, en castellano, las frases a menudo siguen el patrón "sujeto-predicado", pero se requiere cierto conocimiento de la lengua para detectar el patrón. El reconocimiento de patrones se estudia en muchos campos, incluyendo psicología, etología, informática y procesado digital de la señal.

En este proyecto vamos a estudiar unos algoritmos de reconocimiento de patrones basados en el producto escalar, y más particularmente en la desigualdad de Cauchy-Schwarz. Aplicaremos dichos algoritmos en 1D a se nales de audio y en 2D a imágenes digitales. Vamos primero a recordar las nociones de producto escalar y espacio de Hilbert que necesitaremos para abordar la desigualdad de Cauchy-Schwarz y posteriormente la definición de los filtros que permiten llevar a cabo la búsqueda de patrones.

2.1. Producto Escalar

El producto escalar, también conocido como producto interno o producto punto, es una función definida sobre un espacio vectorial cuyo resultado es una magnitud escalar. El nombre espacio escalar se utiliza para denomi-

nar un espacio vectorial real sobre el que se ha definido una operación de producto interior que tiene como resultado un número real (ver [4]).

2.1.1. Definición general

El producto interior o producto escalar de dos vectores en un espacio vectorial es una forma sesquilineal, hermítica y definida positiva, i.e., una operación:

$$< \cdot, \cdot >: VxV \rightarrow K$$

donde V es el espacio vectorial y K es el cuerpo sobre el que está definido, que tiene que cumplir:

$$1) < ax + by, z >= a < x, z > +b < y, z >$$

(lineal en el primer componente)

$$2) < x, y >= \overline{< x, y >}$$

(hermítica)

$$3) < x, x > \geq 0, y < x, x >= 0$$

si y sólo si (definida positiva)

donde x, y, z son vectores arbitrarios,a, b representan escalares cualesquiera y $\overline{< x, y >}$ es el conjugado del complejo $< x, y >$.

Si el cuerpo tiene parte imaginaria nula (v.g.,$\mathcal{R}$), la propiedad de ser sesquilineal se convierte en ser bilineal y el ser hermítica se convierte en ser simétrica.

Un espacio vectorial sobre el cuerpo $\mathcal{R}$ o $\mathcal{C}$ dotado de un producto escalar se denomina espacio prehilbert o espacio prehilbertiano. Si además es completo, se dice que es un espacio de hilbert, y si la dimensión es finita, se dirá que es un espacio euclídeo.

Todo producto escalar induce una norma sobre el espacio en el que está definido, de la siguiente manera:

$$||x|| := \sqrt{< x, x >}$$

A continuación y por completitud a nadimos la noción de Espacio de Hilbert aunque no es necesaria su lectura para la comprensión del resto.

2.2. Espacio de Hilbert

Un espacio de Hilbert es un espacio de producto interior que es completo con respecto a la norma vectorial definida por el producto interior. Los espacios de Hilbert sirven para clarificar, y para generalizar el concepto de extensión de Fourier, ciertas transformaciones lineales tales como la transformación de Fourier, y son de importancia crucial en la formulación matemática de la mecánica cuántica. El espacio de Hilbert y sus propiedades se estudia dentro del análisis funcional.

Cada producto interior $< ., . >$ en un espacio vectorial H, que puede ser real o complejo, da lugar a una norma $||.||$ que se define como:

$$||x|| = \sqrt{< x, x >}$$

Decimos que H es un espacio de Hilbert si es completo con respecto a esta norma. Completo en este contexto significa que cualquier sucesión de Cauchy de elementos del espacio converge a un elemento en el espacio, en el sentido que la norma de las diferencias tiende a cero. Cada espacio de Hilbert es así también un espacio de Banach (pero no viceversa).

Todos los espacios finito-dimensionales con producto interior (tales como el espacio euclídeo con el producto escalar ordinario) son espacios de Hilbert. Esto permite que podamos extrapolar nociones desde los espacios de dimensión finita a los espacios de Hilbert de dimensión infinita (por ejemplo los espacios de funciones). Sin embargo, los ejemplos infinito-dimensionales tienen muchos más usos. Estos usos incluyen:

La teoría de las representaciones del grupo unitarias. La teoría de procesos estocásticos cuadrado integrables. La teoría en espacios de Hilbert de ecuaciones diferenciales parciales, en particular formulaciones del problema de Dirichlet. Análisis espectral de funciones, incluyendo teorías de wavelets.

Formulaciones matemáticas de la mecánica cuántica. El producto interior permite que uno adopte una visión 'geométrica' y que utilice el lenguaje geométrico familiar de los espacios de dimensión finita. De todos los espacios vectoriales topológicos infinito-dimensionales, los espacios de Hilbert son los de 'mejor comportamiento' y los más cercanos a los espacios finito-dimensionales.

Los elementos de un espacio de Hilbert abstracto a veces se llaman 'vectores'. En las aplicaciones, son típicamente sucesiones de números complejos o de funciones. En mecánica cuántica por ejemplo, un conjunto físico es descrito por un espacio complejo de Hilbert que contenga las 'funciones de ondas' para los estados posibles del conjunto.

Una de las metas del análisis de Fourier es facilitar un método para escribir una función dada como la suma (posiblemente infinita) de múltiplos de funciones bajas dadas. Este problema se puede estudiar de manera abstracta en los espacios de Hilbert: cada espacio de Hilbert tiene una base ortonormal, y cada elemento del espacio de Hilbert se puede escribir en una manera única como suma de múltiplos de estos elementos bajos.

Los espacios de Hilbert fueron nombrados así por David Hilbert, que los estudió en el contexto de las ecuaciones integrales. El origen de la designación, aunque es confuso, fue utilizado ya por Hermann Weyl en su famoso libro la teoría de grupos y la mecánica cuántica publicado en 1931. John von Neumann fue quizás el matemático que más claramente reconoció su importancia (ver [7]).

2.3. Desigualdad de Cauchy-Schwarz

La desigualdad de Cauchy-Schwarz, también conocida como desigualdad de Schwarz o la desigualdad de Cauchy-Bunyakovski-Schwarz es una desigualdad que establece que si x e y son elementos de un espacio prehilbertiano se verifica:

$$| < x, y > |^2 \leq < x, x > \cdot < y, y >$$

Y se da la igualdad si y solo si x y y son linealmente dependientes, es decir

x=ay para un escalar a.

2.4. Algoritmo de Reconocimiento de Patrones basado en la Desigualdad de Cauchy-Schwarz

Utilizando la desigualdad de Cauchy-Schwarz estudiada en la sección 2.3 podemos definir un algoritmo de reconocimiento de patrones mediante el siguiente filtro. Supongamos que tenemos el patrón de búsqueda y. Entonces lo que tenemos que hacer es compararlo con los posibles vectores x mediante el siguiente filtro:

$$\frac{< x, y >}{\sqrt{< x, x >} \cdot \sqrt{< y, y >}}. \tag{2.1}$$

Un valor del filtro igual a 1 indicará que el vector x es igual al patrón. Si el valor es distinto a 1, entonces será menor. Lo interesante es que cuanto más próximo esté el valor a 1 más se parecerán x e y.

Este filtro también es común verlo para vectores normalizados previamente, es decir, $||y|| = 1$, $||x|| = 1$. En este caso el filtro consiste simplemente en realizar el producto escalar de los vectores:

$$< x, y > . \tag{2.2}$$

Es interesante recordar que

$$< x, y >= ||x||||y||cos(\theta), \tag{2.3}$$

donde θ es el ángulo formado por los vectores x e y. Como tanto la función norma como la función coseno son funciones continuas, esto implica que peque nas variaciones en y o en x darán peque nas variaciones en el valor del filtro. Esta propiedad será crucial para obtener un algoritmo estable frente a peque nas transformaciones bien del patrón o bien del espectro de vectores donde se realiza la búsqueda.

Capítulo 3

Definición del Algoritmo en 1D. Aplicación al procesado de señales de audio

En este capítulo vamos a presentar un algoritmo de reconocimiento de patrones para señales uno dimensionales. El algoritmo está basado en el filtro dado en las fórmulas (2.1)-(2.2). Vamos a explicar el algoritmo paso a paso con la ayuda de la función de la Figura 3.1.

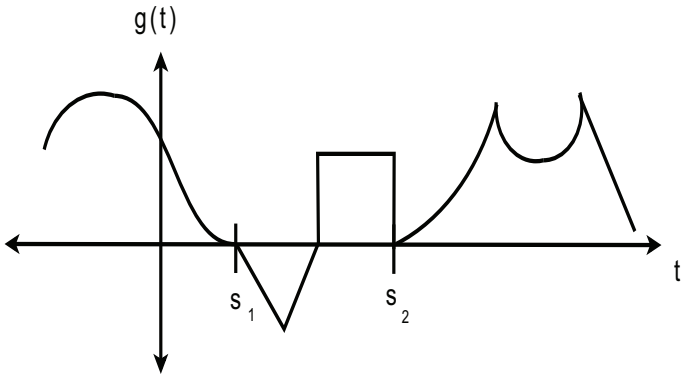


Figura 3.1: Señal continua

Se trata de una función continua que vamos a discretizar para pasar a trabajar en un espacio finito dimensional. Igualmente tenemos también un patrón a buscar dentro de la señal, el cual es a su vez una señal continua,

como muestra la Figura 3.2, que tendremos que discretizar. Supongamos pues que la función g de manera discreta viene representada por el vector n-dimensional v, y que el patrón f lo está por el vector m-dimensional y.

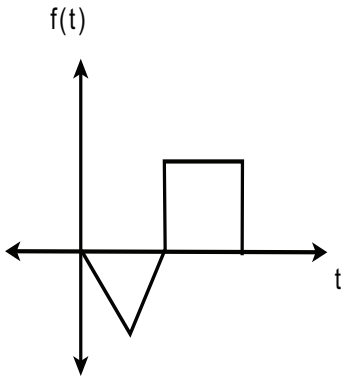


Figura 3.2: Señal continua que utilizamos de patrón de búsqueda

El primer paso del algoritmo será normalizar y haciéndolo así unitario. Podemos suponer de esta manera que y tiene norma 1. El segundo paso será ir tomando muestras del vector v de tamaño $m < n$. Podemos tomar $n - m + 1$ muestras. LLamaremos x a una de estas muestras. El camino a seguir consistirá en normalizar x para hacerlo unitario y pasar entonces el filtro con entradas los vectores x e y. Este proceso deberá realizarse con cada una de las posibles muestras x contenidas en el vector v, comparando a cada paso los valores del filtro, quedándonos siempre con la posición inicial de la muestra que da un valor más aproximado de 1. Ésa será la posición donde el algoritmo detecta el patrón dentro de la señal. En el caso del ejemplo esta posición coincidirá con el punto S_1 como puede verse en la Figura 3.1.

Esquemáticamente podemos expresar los diferentes pasos del algoritmo de la siguiente manera

Algoritmo 1D

- Obtener las versiones discretizadas de la señal y del template
- Normalizar el template y
- Para cada una de las posibles muestras x del tamaño del template contenidas en la señal hacer:

- Normalizar x
- Calcular el filtro $< x, y >$
- Comparar con el valor anterior del filtro. Si es mayor guardar la posición de x en la variable *posicion*

- La variable *posicion* contiene la localización del template

3.1. Aplicación a las señales de audio

Las señales de audio son un caso particular de señales uno dimensionales a las que les podemos aplicar el algoritmo anterior. Pensar por ejemplo en una pista de música en la cual queremos localizar una parte concreta. Esta parte concreta a localizar será nuestro template en el algoritmo de búsqueda.

Para este fin hemos desarrollado una interfaz gráfica que permite al usuario ejecutar los programas realizados en lenguaje Matlab de una manera sencilla. En la Figura 3.3 se muestra la interfaz gráfica creada. En ella vemos que aparece las siguientes opciones: lectura de una señal; audición de la señal; adición de ruido a la señal; lectura de un template; audición del template; aplicación del algoritmo de búsqueda.

Vamos a llevar a cabo una búsqueda en concreto. Para ello leemos la señal 'heli.wav' que contiene el sonido generado por un helicóptero (ver Figura 3.4).

En primer lugar realizaremos el experimento sin adicionar ruido. Con lo que el segundo paso consistirá en leer el template de búsqueda, en nuestro caso la señal 'template_heli.wav' (ver Figura 3.5).

A continuación sólo tenemos que picar en el botón *Aplicar* para ejecutar el algoritmo. El resultado de la búsqueda aparece en la Figura 3.6, donde se ha marcado en rojo la parte de la señal buscada. La posición exacta y el valor numérico exacto del filtro se ofrecen justo encima de la señal. Que el filtro tenga valor 1 nos indica que el template está contenido tal cual en la señal.

Para testar la robustez del algoritmo vamos a adicionar a la señal 'heli.wav' ruido blanco antes de realizar la búsqueda. Empezamos con un ruido

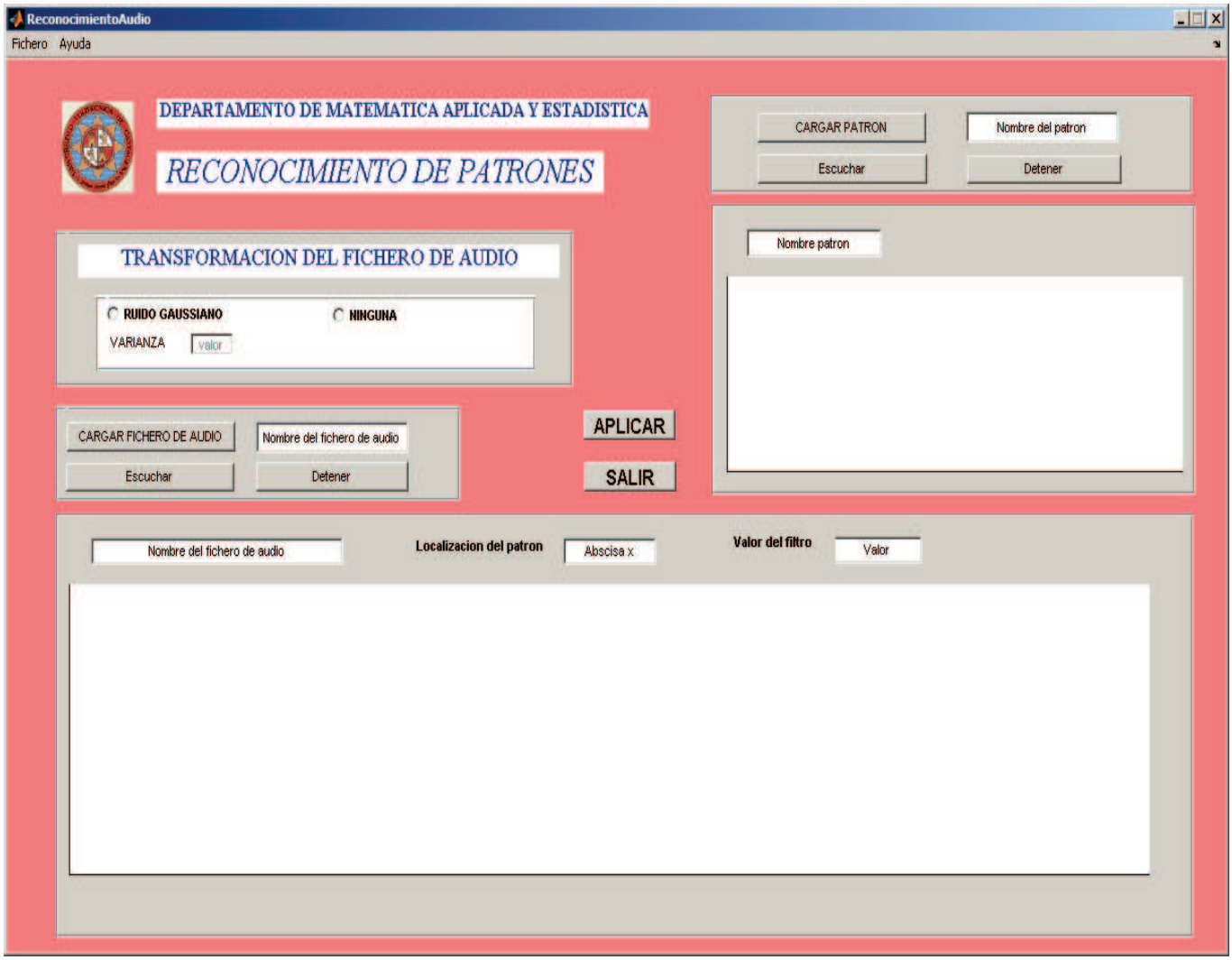


Figura 3.3: Interfaz para reconocimiento de patrones en señales de audio

de varianza $\sigma = 0{,}01$ y como podemos comprobar en la Figura 3.7 la búsqueda se realiza con éxito.

El siguiente paso será adicionar ruido de mayor varianza hasta que el algoritmo deje de localizar el patrón con exactitud. En la Figura 3.8 vemos como el algoritmo ya ha sido afectado por una cantidad excesiva de ruido.

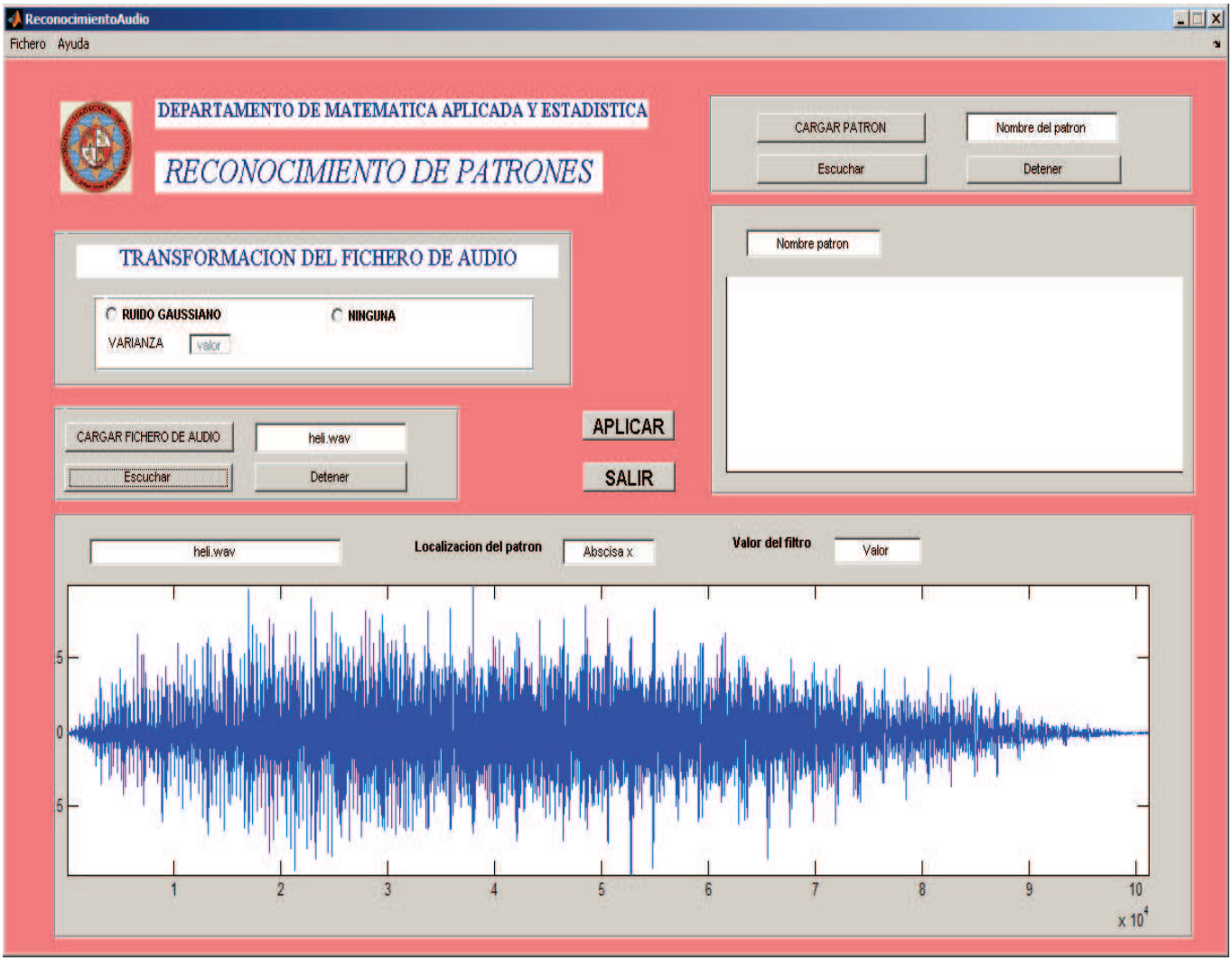


Figura 3.4: Lectura de la señal de audio 'heli.wav'

Como conclusión de este apartado podemos decir que el algoritmo no es muy sensitivo a cantidades moderadas de ruido. La razón para que esto ocurra nos la ofrece la fórmula 2.3, donde se ve que el filtro es una función continua.

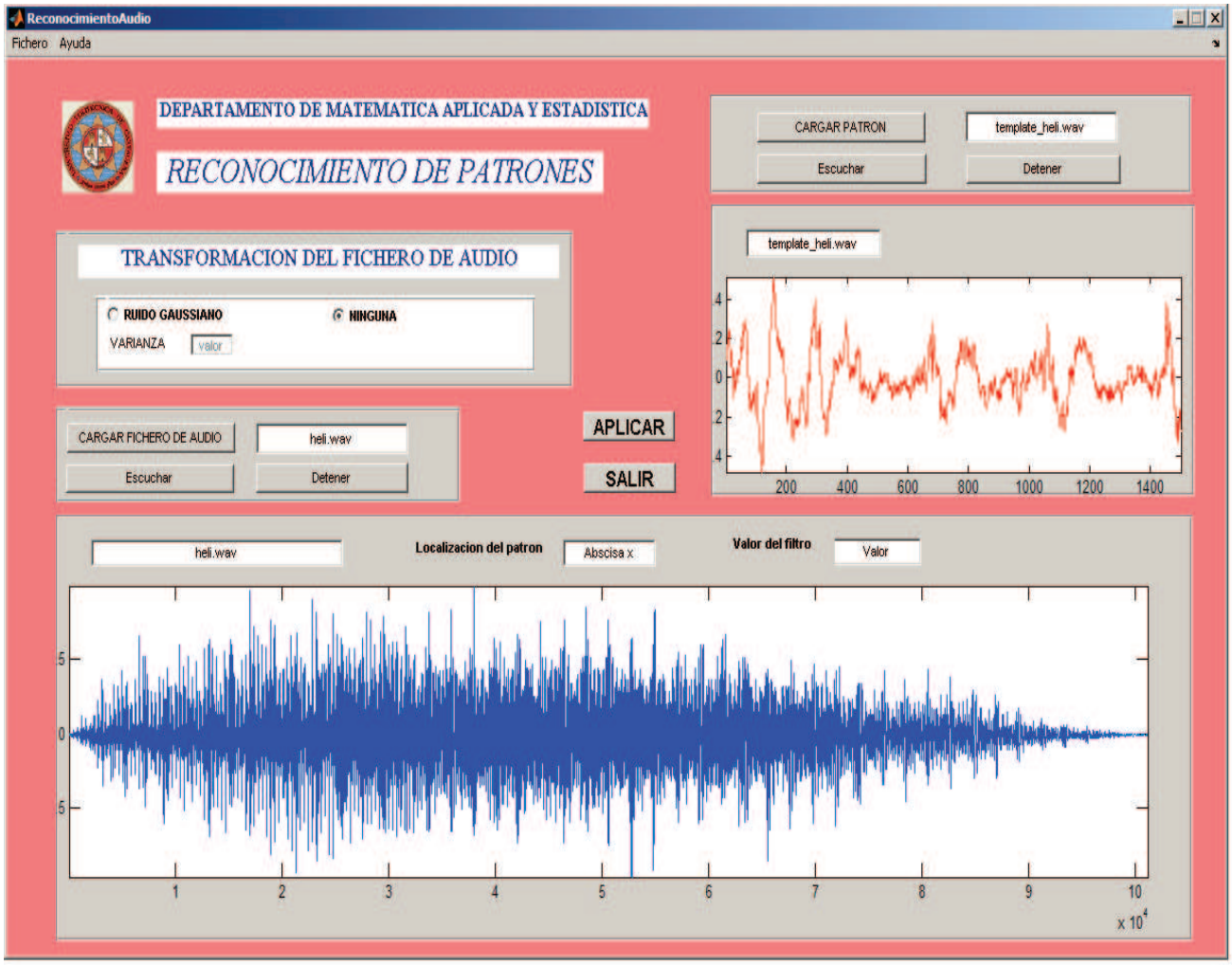


Figura 3.5: Lectura del patrón de búsqueda para la señal de audio 'heli.wav'

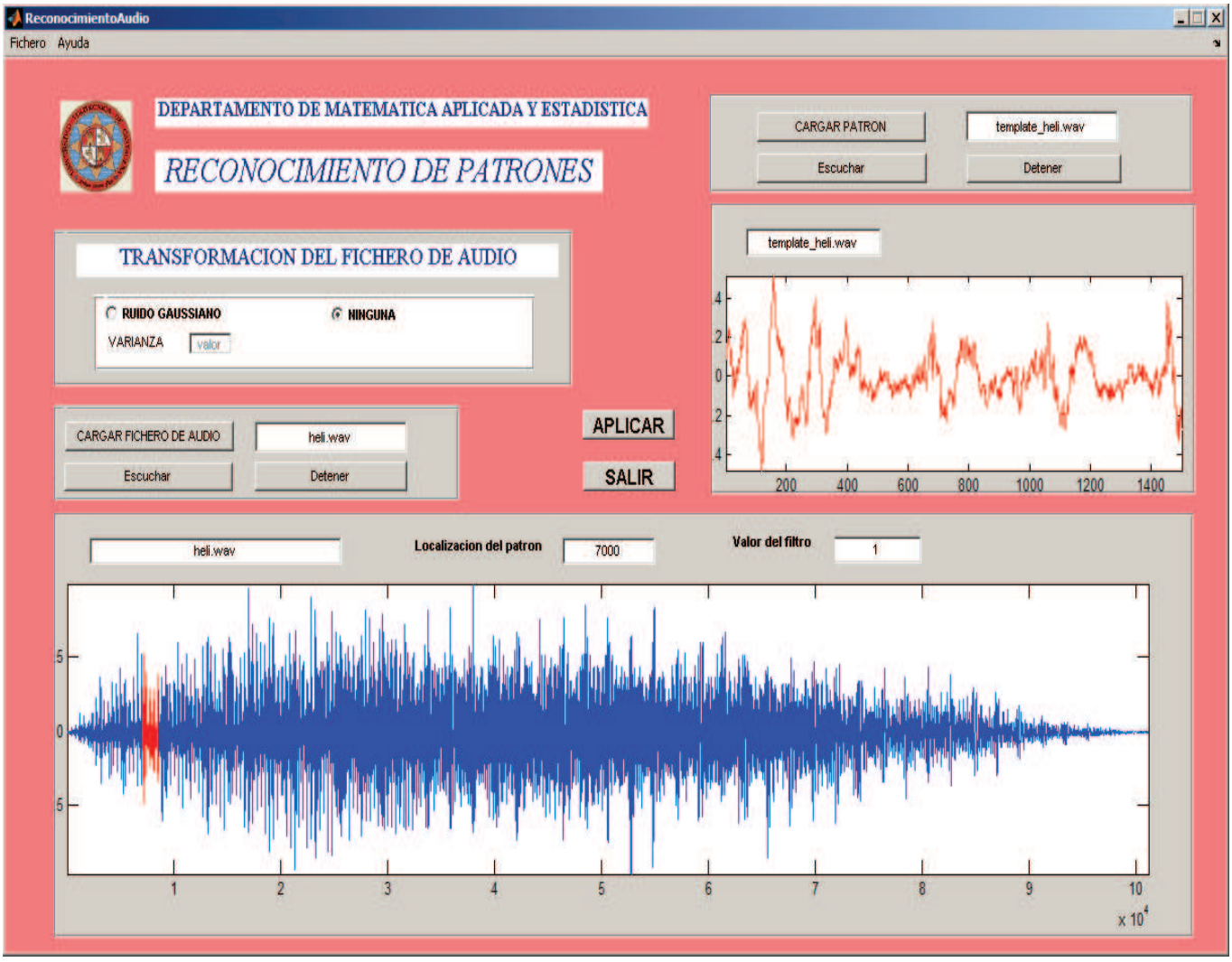


Figura 3.6: Búsqueda del patrón en la señal de audio 'heli.wav'

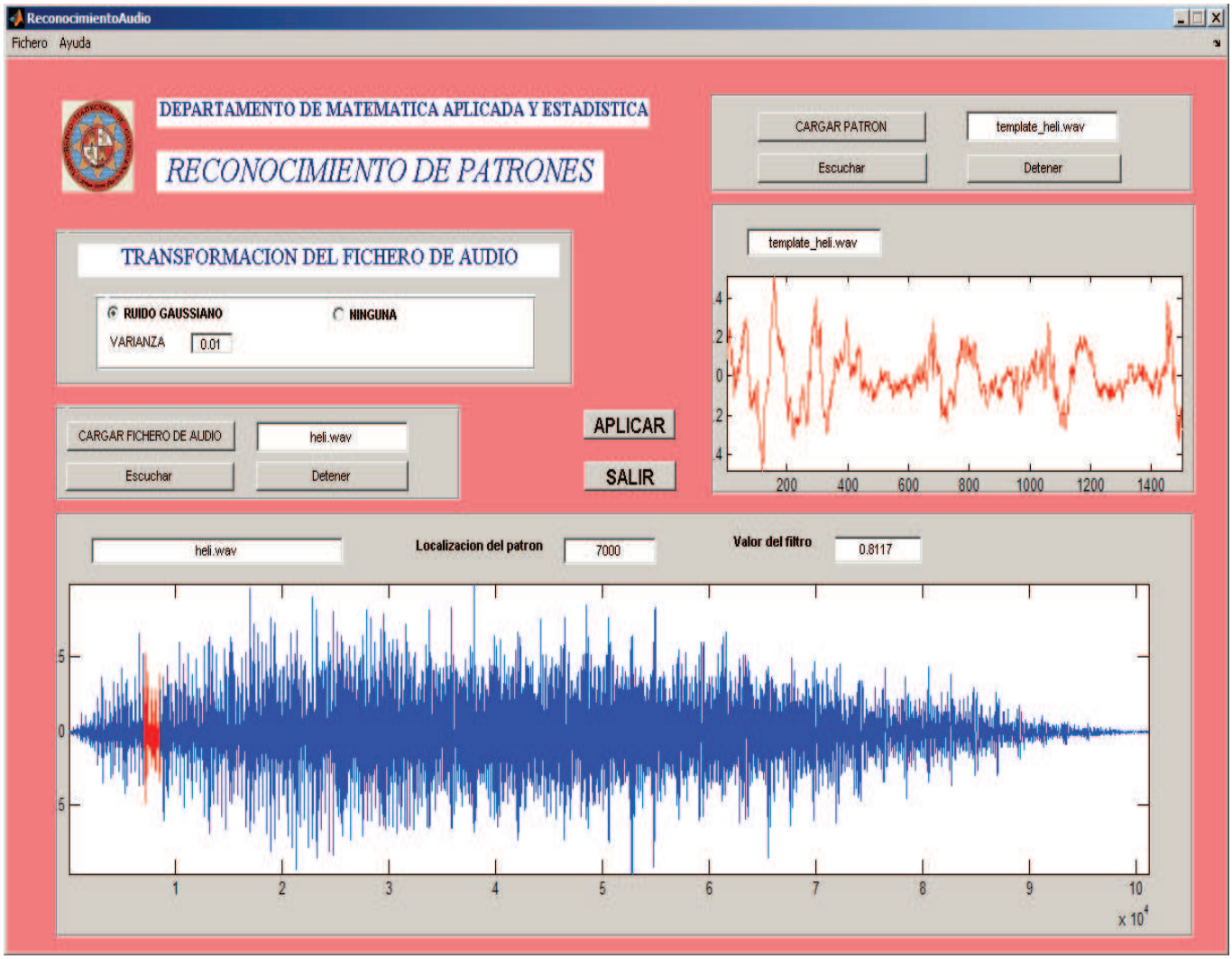


Figura 3.7: Búsqueda del patrón en la señal de audio 'heli.wav' alterada con ruido gausiano de varianza $\sigma = 0{,}01$

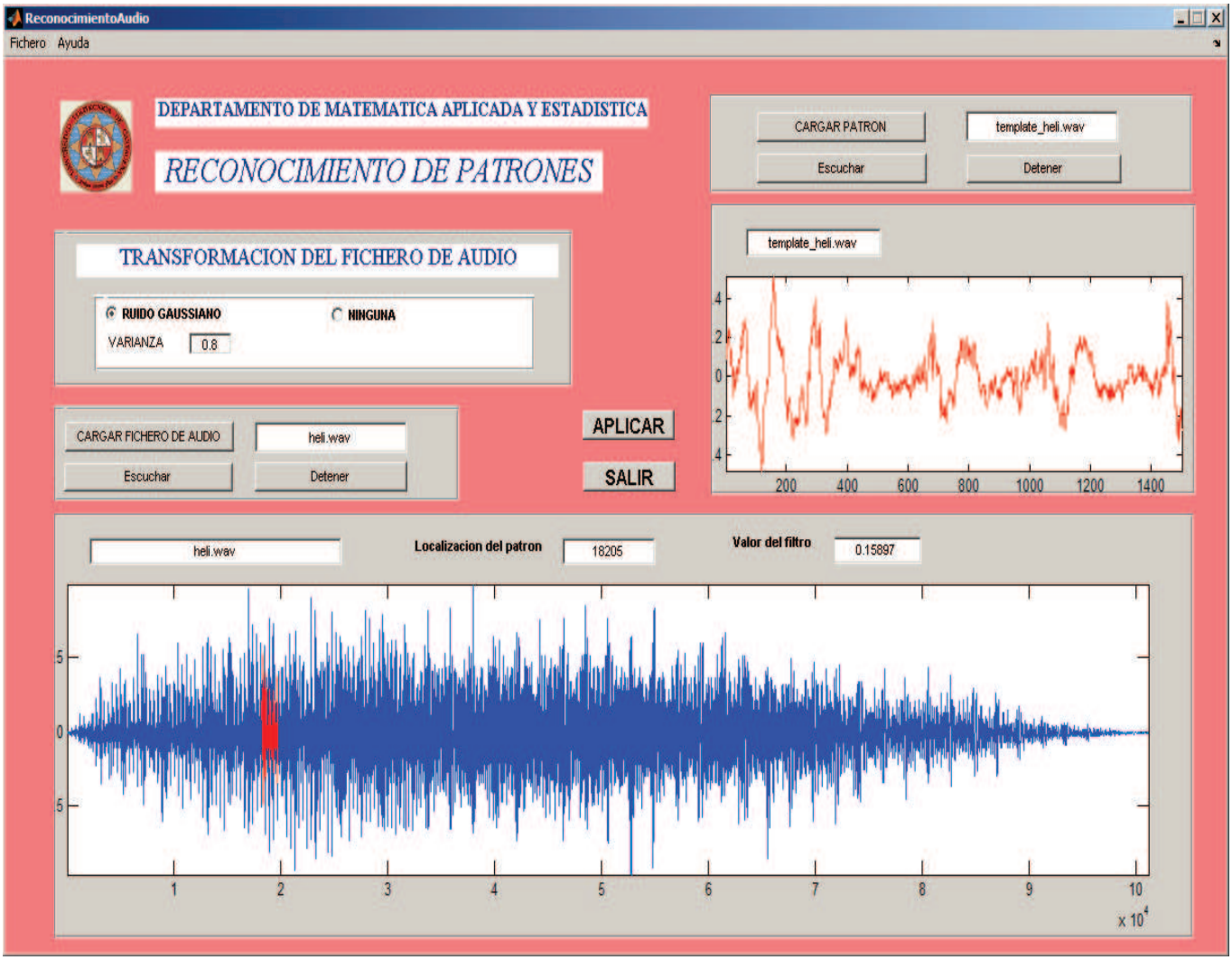


Figura 3.8: Búsqueda del patrón en la señal de audio 'heli.wav' alterada con ruido gausiano de varianza $\sigma = 0{,}8$

Capítulo 4

Definición del Algoritmo en 2D. Aplicación al procesado de imágenes digitales

En este capítulo vamos a presentar un algoritmo de reconocimiento de patrones para señales dos dimensionales. El algoritmo está basado en el filtro dado en las fórmulas (2.1)-(2.2). Vamos a explicar el algoritmo paso a paso con la ayuda de la Figura 4.1.

Supongamos que tenemos discretizada la imagen de manera que estamos trabajando en un espacio finito dimensional. Igualmente tenemos también un patrón a buscar dentro de la señal, el cual es a su vez una imagen discretizada, llamemosla V, como muestra la Figura 4.1.

El primer paso del algoritmo será normalizar el patrón V haciéndolo así unitario. Para ello utilizaremos la fórmula:

$$\bar{V}_{ij} = \frac{V_{ij}}{\sqrt{\sum_i \sum_j V_{ij}^2}}. \tag{4.1}$$

Podemos suponer de esta manera que V tiene norma 1. El segundo paso será ir tomando muestras de la imagen del mismo tamaño que V. LLamemos U a una de estas muestras. El camino a seguir consistirá en normalizar U para hacerlo unitario y pasar entonces el filtro con entradas las matrices U y V, las cuales previamente las hemos dispuesto en forma de vector. Este

Figura 4.1: Explicación gráfica del algoritmo de patrones en 2D

proceso deberá realizarse con cada una de las posibles muestras U contenidas en la imagen, comparando a cada paso los valores del filtro, quedándonos siempre con la posición inicial de la muestra que da un valor más aproximado de 1. Ésa será la posición donde el algoritmo detecta el patrón dentro de la señal. En el caso del ejemplo de la Figura 4.1 el valor del filtro será exactamente 1 para la muestra contenida en la matriz U_2.

Esquemáticamente podemos expresar los diferentes pasos del algoritmo de la siguiente manera:

Algoritmo 2D

- Obtener las versiones discretizadas de la imagen y del template
- Normalizar el template V
- Para cada una de las posibles muestras U del tamaño del template contenidas en la imagen hacer:
  - Normalizar U
  - Calcular el filtro $< U, V >$

    - Comparar con el valor anterior del filtro. Si es mayor guardar la posición de U en la variable *posicion*

- La variable *posicion* contiene la localización del template

4.1. Aplicación a las imágenes digitales

Hemos desarrollado una interfaz gráfica que permite al usuario ejecutar los programas realizados en lenguaje Matlab que llevan a cabo el reconocimiento de patrones descrito de una manera sencilla.

En la Figura 4.2 se muestra la interfaz gráfica creada. En ella vemos que en la parte superior derecha se encuentran las opciones para cargar una imagen y un patrón a buscar. En la parte izquierda aparecen las siguientes opciones para transformar una imagen digital: rotación; filtro de suavidad con sus respectivas modalidades de mediana y vecindad; el ruido con sus diferentes modalidades: **gaussiano**, **sal y pimienta** y **multiplicativo**; **compresión en** $JPEG$; **ecualización**; todo ello, para estudiar la robustez del algoritmo.

Una vez marcadas los campos elegidos se ejecuta el programa pinchando en el botón *Aplicar* en la parte inferior derecha de la interfaz.

Vamos a llevar a cabo una búsqueda en concreto. Para ello leemos la imagen 'ciclo.png' que contiene una imagen real del autor del proyecto en su tiempo libre dirigiendo una clase de ciclo indoor (ver Figura 4.3). En primer lugar cargaremos la imagen $'ciclo.png'$ incorporando la ruta para que el programa la localice y la imagen sea cargada sin problemas.

El segundo paso consistirá en leer el template de búsqueda, en nuestro caso la muestra de la imagen 'template-ciclo.png' (ver Figura 4.4) añadiéndole la ruta del directorio donde tenemos la imagen.

A continuación sólo tenemos que picar sobre cualquiera de las distintas transformaciones añadiéndole los valores que estimemos oportunos en cada caso. Por ejemplo, si queremos aplicar la ***Rotación*** tenemos que meter un ángulo de los grados en los que queramos rotar la imagen.

Si queremos aplicar el ***Filtro de Suavidad*** tendremos que seleccionar uno de los 2 tipos que tiene, ya sea el ***Filtro de Vecindad*** ó el ***Filtro de Mediana*** y para cualquiera de los 2 casos hay que introducir un valor para

Figura 4.2: Interfaz para reconocimiento de patrones en imágenes digitales

N y otro para M que son las dimensiones de la máscara.

Si queremos aplicar ***Ruido*** tendremos que seleccionar uno de los 3 tipos que tiene, ya sea el ***Gaussiano***, **el *Sal y Pimienta*** o el ***Multiplicativo***. Para el ***Ruido Gaussiano*** hay que introducir un valor para la $Varianza$ y

Figura 4.3: Lectura de la imagen digital *'ciclo.png'*

otro para la *Media*. Para el ***Ruido Sal y Pimienta*** hay que introducir un valor para la *Densidad*. Para el ***Ruido Multiplicativo*** hay que introducir un valor para la *Varianza*.

Si queremos aplicar un ***Factor de Compresión del tipo JPEG*** a la

Figura 4.4: Lectura del patrón de búsqueda para la imagen digital $'template - ciclo.png'$

imagen hay que introducir un valor para determinar la *Calidad*. Veremos en el Capítulo 5 los distintos tipos de compresión digital y veremos que el formato $JPEG$ es muy eficiente y que en la actualidad es uno de los métodos más utilizado por los usuarios.

Y para terminar también tenemos la posibilidad de aplicar una ***Ecualización*** a una imagen digital. Todos estos ejemplos con imágenes,los veremos detalladamente en el capítulo 5, explicando cada transformación y los respectivos resultados de los experimentos.

El resultado de aplicar este algoritmo sin seleccionar ninguna transformación sobre la imagen es reconocer el template ó patrón de búsqueda dentro de la propia imagen mediante un cuadrado de color rojo, como se puede ver en la figura 4.5.

Figura 4.5: Resultado de aplicar el algoritmo sin seleccionar ninguna transformación

Capítulo 5

Tipos de transformaciones de imágenes

5.1. Rotación

Es una transformación que consiste en rotar una imagen un determinado ángulo.

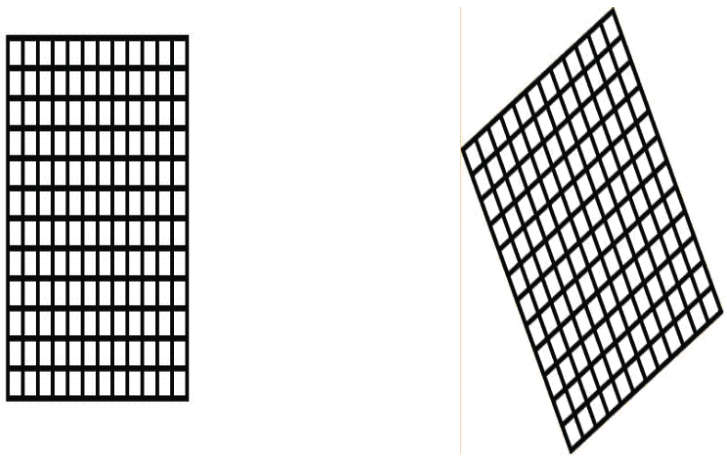


Figura 5.1: Efecto de la rotación

Si cada cuadrito es un píxel esta claro que los píxeles de la imagen girada no coincidirán con la rejilla que genera una imagen nueva, o visto de

otra manera, cuando estemos hallando a que píxel corresponde el (x, y) de la imagen girada nos dará un (x', y') donde x' e y' no serán necesariamente enteros. Esto nos obliga a tener que hacer una interpolación.

Veamos que aspecto tiene la ecuación matricial de un giro:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix},$$

para un giro de ángulo alfa.

En teoría se puede girar una imagen un ángulo alfa y luego menos alfa para volver a dar la imagen original. Esto es simplemente usar la misma matriz en la que los términos sin cambian de signo, ya que:

$$\sin(-\alpha) = -\sin(\alpha)$$

Esto también quiere decir que la matriz resultante de hacer esa operación es la inversa de la original. En la realidad si hacemos el primer giro, interpolamos, y esto puede conllevar una pérdida de información que nos impida recuperar la imagen original exactamente.

5.1.1. Comandos

Los comandos utilizados en Matlab para rotar una imagen son los siguientes:

```
>> a = imread (filename);                          % lee la imagen
>> variable = imrotate(x,30,'nearest','crop');     % gira la imagen
>> imshow (variable, map)                          % muestra la imagen
```

5.1.2. Experimentos y resultados

Para realizar todos los experimentos tomaremos la imagen 'camera2.png' que se puede ver en la figura 5.2 y y el patrón a buscar en la figura 5.3.

Figura 5.2: Imagen elegida a la que le vamos a aplicar las transformaciones

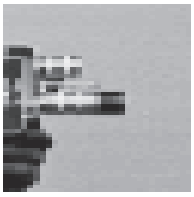


Figura 5.3: Template elegido al que le vamos a aplicar las transformaciones

A continuación, pondremos un ejemplo del cálculo de una rotación de un ángulo de 10 grados como vemos en la figura 5.4 y veremos que el algoritmo ha funcionado eficientemente porque, aunque rotada la imagen, ha encontrado el patrón como vemos en la figura 5.5.

A continuación, pondremos un ejemplo para mostrar que el algoritmo es robusto frente a rotaciones. Para ello hemos rotado la imagen un ángulo de 30 grados como vemos en la figura 5.6 y el algoritmo sigue reconociendo perfectamente el patrón seleccionado previamente dentro de la imagen como vemos en la figura 5.7.

Figura 5.4: Imagen rotada con un ángulo de 10 grados

Figura 5.5: Reconocimiento del patrón dentro de la imagen rotada 10 grados

Figura 5.6: Imagen rotada con un ángulo de 30 grados

Figura 5.7: Reconocimiento del patrón dentro de la imagen rotada 30 grados

5.2. Filtros de suavidad

Las operaciones de suavizado se utilizan para disminuir los efectos negativos que se pueden presentar en una imagen digital como consecuencia de un sistema de muestreo pobre o del canal de transmisión. Por ejemplo ruido.

5.2.1. Filtro de Mediana

Los filtros de suavizado lineales o filtros paso bajo tienden a difuminar los ejes a causa de que las altas frecuencias de una imagen son atenuadas. La visión humana es muy sensible a esta información de alta frecuencia. La preservación y el posible realce de este detalle es muy importante al filtrar. Cuando el objetivo es más la reducción del ruido que el difuminado, el empleo de los filtros de mediana representan una posibilidad alternativa.

A menudo, las imágenes digitales se corrompen con ruido durante la transmisión o en otras partes del sistema. Esto se ve a menudo en las imágenes convertidas a digital de una señal de la televisión. Usando técnicas del filtrado de ruido, el ruido puede ser suprimido y la imagen corrompida se puede restaurar a un nivel aceptable. En aplicaciones de ingeniería eléctrica, el ruido se elimina comúnmente con un filtro paso bajo. El filtrado paso bajo es satisfactorio para quitar el ruido gaussiano pero no para el ruido impulsivo. Una imagen corrupta por ruido impulsivo tiene varios píxeles que tienen intensidades visiblemente incorrectas como 0 o 255. Hacer un filtrado paso bajo alterarán estas señales con los valores extremos sobre la vecindad del píxel. Un método mucho más eficaz para eliminar el ruido impulsivo es el filtrado de mediana.

En el filtrado de mediana, el nivel de gris de cada píxel se reemplaza por la mediana de los niveles de gris en un entorno de este píxel, en lugar de por la media. Recordar que la mediana M de un conjunto de valores es tal que la mitad de los valores del conjunto son menores que M y la mitad de los valores mayores que M, es decir en un conjunto ordenado de mayor a menor o viceversa, sería el valor de la posición central.

El filtro de la mediana no puede ser calculado con una máscara de convolución, ya que es un filtro no lineal. Podemos ver como este tipo de filtro elimina totalmente el punto que tenia un valor muy diferente al resto de

sus vecinos. Como se selecciona el valor de centro, el filtrado de mediana consiste en forzar que puntos con intensidades muy distintas se asemejen más a sus vecinos, por lo que observamos que el filtro de mediana es muy efectivo para eliminar píxeles cuyo valor es muy diferente del resto de sus vecinos, como por ejemplo eliminando ruido de la imagen.

Al implementar el filtro de mediana encontramos el mismo problema de bordes que teníamos en la convolución: cuándo la ventana de filtrado está centrada en el píxel (0,0), el filtrado lineal (media) da un mejor resultado a la hora de eliminar ruido gaussiano, mientras que el filtrado no lineal (mediana) es más adecuado a la hora de eliminar ruido impulsivo.

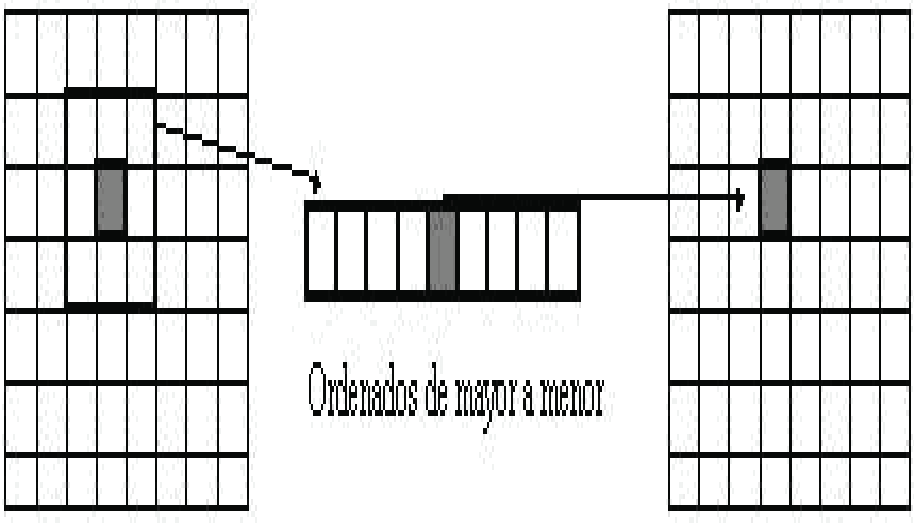


Figura 5.8: Aplicación del filtro de mediana

5.2.2. Filtro de Vecindad

Dada una imagen $f(x, y)$ de tamaño nxm, para aplicar un filtro a la imagen es necesario definir una matriz que contendrá los coeficientes del filtro, lo que a su vez define los píxeles del entorno que serán utilizados como argumento del filtro que alterará el valor del píxel. A esta matriz se le denomina máscara con dimensión [m, n]. El valor del nivel de gris de la imagen suavizada $g(x, y)$ en el punto (x, y) se obtiene promediando valores de nivel de gris de los puntos de f contenidos en una cierta vecindad de (x, y).

$$g(x, y) = \frac{1}{M} \sum_{(n,m) \in S} f(n, m)$$

donde $x, y = 0, 1, ..., N-1$. S es el conjunto de coordenadas de los puntos vecinos a (x, y), incluyendo el propio (x, y), y M es el número de puntos de la vecindad. Por ejemplo, imaginemos la subimagen y la máscara siguientes:

$f(x-1, y-1)$	$f(x, y-1)$	$f(x+1, y-1)$
$f(x-1, y)$	$f(x, y)$	$f(x+1, y)$
$f(x-1, y+1)$	$f(x, y+1)$	$f(x+1, y+1)$

Tabla 5.1: Valores de los pixeles de la subimagen

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Tabla 5.2: Valores de la máscara del filtro

y que queremos reemplazar el valor de $f(x, y)$ por el promedio de los puntos en una región de tamaño 3x3 centrada en (x, y), es decir, queremos asignar el valor promedio a $g(x, y)$:

$$\begin{aligned} g(x, y) &= 1/9(f(x-1, y-1) + f(x-1, y) + f(x+1, y-1) \\ &+ f(x-1, y) + f(x, y) + f(x+1, y) \\ &+ f(x-1, y-1) + f(x, y+1) + f(x+1, y+1)) \end{aligned}$$

Esta operación se puede realizar de forma general centrando la máscara en (x, y) y multiplicando cada punto debajo de la máscara por el correspondiente coeficiente de la matriz y sumando el resultado.

El problema del filtro de vecindad es que aparece la difuminación de bordes. Por ejemplo a un filtro de vecindad es preciso proporcionarle dos implementaciones de imagen: la que tiene la información y aquella donde se desea obtener el resultado, ya que es posible que el usuario desee implementaciones distintas en ambas imágenes. En ciertos casos es posible devolver el resultado en una de las imágenes de entrada, ésta en general es la opción más eficiente si solo es importante el resultado y hemos adoptado como política que en los algoritmos donde es posible devolver el resultado en uno de los parámetros de entrada, se usa esta opción por defecto. Es responsabilidad del usuario mantener una copia de la entrada de ser necesario. El caso del filtro de vecindad es un típico ejemplo donde es preciso devolver el resultado en otra imagen ya que es naturalmente destructivo
(no puedo calcular correctamente el valor de la vecindad de un pixel en la imagen, si previamente modifiqué el pixel anterior). En este caso el usuario debe proporcionar dos imágenes: una de donde leer la información que se devuelve inalterada y otra donde se escribe el resultado.

5.2.3. Comandos

Filtro de mediana:

```
a=imread(filename);                % lee la imagen
[na,ma,ban]=size(a);               % calcula el tamaño de la imagen
for k=1:lg                         % bucle que recorre las bandas
a(:,:,k)=medfilt2(a(:,:,k),[N,M]); % aplica el filtro en cada banda
end
```

Filtro de vecindad:

```
 mascara=1/N/M*ones(N,M);      % máscara de filtro
 a=conv2(a,mascara,'same');    % aplicación del filtro de convolución
```

5.2.4. Experimentos y resultados

Vamos a comenzar con la primera transformación dentro de los filtros de suavidad que es el filtro de mediana. A continuación, pondremos un ejemplo

del cálculo de la aplicación de un filtro de mediana poniendo a N y M el valor de 5, como vemos en la figura 5.9 el cálculo ha sido hallado eficientemente ya que aplicando el filtro de mediana a la imagen, el algoritmo ha encontrado el patrón a buscar, como vemos en la figura 5.10.

Figura 5.9: Imagen filtrada con el filtro de mediana $N = M = 5$

A continuación, pondremos un ejemplo donde comprobamos que el algoritmo es preciso incluso con valores elevados de N y M. En la figura 5.11 vemos el resultado de aplicar el filtro de mediana con $N = M = 20$ a la imagen y en la figura 5.12 mostramos el resultado del algoritmo de búsqueda. Hemos testado así la robustez frente a este tipo de transformación.

Vamos a continuar con la segunda transformación dentro de los filtros de suavidad que es el filtro de vecindad. Aplicamos a la imagen un filtro de vecindad con N y M iguales a 5. El resultado lo vemos en la figura 5.13. A continuación aplicamos el algoritmo de reconocimiento de patrones y observamos que la salida es satisfactoria, como vemos en la figura 5.14.

A continuación, le aplicamos a la imagen el filtro de vecindad con los valores de N y M de 20 (ver la figura 5.15), y comprobamos que el algoritmo reconoce perfectamente el patrón (ver la figura 5.16). Así una vez más

Figura 5.10: Reconocimiento del patrón dentro de la imagen filtrada con un filtro de mediana $N = M = 5$

hemos podido testar la robustez.

5.3. Ruido

Se denota por ruido cualquier entidad en las imágenes, datos o resultados intermedios que no son interesantes para la computación que se pretende llevar a cabo, que no se corresponde exactamente con la realidad.

El ruido se debe, la mayoría de las veces al equipo electrónico utilizado en la captación de las imágenes (ruido de cuantificación de la imagen, efecto de niebla en la imagen... etc) y al ruido añadido en los tramos de transmisión (posibles interferencias o errores al transmitir los bits de información). Vamos a distinguir tres clases diferentes de ruido: gaussiano, sal y pimienta y multiplicativo.

Figura 5.11: Imagen filtrada con el filtro de mediana con valores altos de $N = M = 20$

5.3.1. Ruido Gaussiano

Se caracteriza por tener un espectro de energía constante para todas las frecuencias. Cuando se presenta este problema, el valor exacto de cualquier píxel es diferente cada vez que se captura la misma imagen. Este efecto, suma o resta un determinado valor al nivel de gris real y es independiente de los valores que toma la imagen.

El ruido gaussiano tiene un efecto general en toda la imagen, es decir, la intensidad de cada píxel de la imagen se ve alterada en cierta medida con respecto a la intensidad en la imagen original. Por el contrario, se observa que el ruido impulsivo tiene un efecto más extremo sobre un subconjunto del total de píxeles de la imagen. Un tanto por ciento de los píxeles de la imagen toman arbitrariamente el valor extremo 0 o 255.

Una forma de eliminar el ruido de una imagen es mediante el suavizado de imágenes. Dicho de otro modo, el filtrado paso bajo se emplea no sólo para el suavizado de imágenes, sino también para la eliminación de ruido. De hecho, el filtrado paso bajo es una manera efectiva de reducir el ruido

Figura 5.12: Reconocimiento del patrón dentro de la imagen filtrada con un filtro de mediana $N = M = 20$

gaussiano en una imagen, mientras que no es tan efectivo con el ruido impulsivo. Como promediar reduce los valores extremos de la vecindad del píxel, hacer un filtro de media tiende a reducir el contraste de las imágenes, pues los valores extremos, altos y bajos, son cambiados por valores medios.

El problema con la utilización de filtros paso bajo para eliminar el ruido de imágenes consiste en que los bordes de los objetos se vuelven borrosos. Los bordes contienen una cantidad enorme de información de una imagen. Filtrando el ruido impulsivo de una imagen, el filtrado de mediana puede ser una mejor opción. Los filtros de mediana hacen un mejor trabajo conservando los bordes.

5.3.2. Ruido Sal y Pimienta ó Impulsivo

Se caracteriza por la aparición de píxeles con valores arbitrarios normalmente detectables porque se diferencian mucho de sus vecinos más próximos. La distribución viene dada por:

Figura 5.13: Imagen filtrada con el filtro de vecindad

$$g(x,y) = \begin{cases} 0 & \text{si} \quad r(x,y) < p/2 \\ L-1 & \text{si} \quad p/2 \leq r(x,y) < p \\ f(x,y) & \text{si} \quad r(x,y) \geq p \end{cases} \tag{5.1}$$

donde L es el número de niveles de grises, $r(x,y)$ es un número aleatorio con distribución uniforme en $[0,1)$ y p es la probabilidad de ocurrencia del ruido aleatorio, es decir, el porcentaje de puntos de la imagen que se verán afectados por el ruido impulsivo del total de puntos de la imagen.

5.3.3. Ruido Multiplicativo

El ruido multiplicativo ó speckle es el patrón de interferencia que se forma cuando una imagen se obtiene a partir de la iluminación de un medio con una radiación, el cual degrada significativamente la calidad de la imagen, aumentando de esta forma la dificultad de discriminar detalles finos en las imágenes durante un examen de diagnostico. También dificulta el procesado de las imágenes, tales como la segmentación y la detección de bordes.

Figura 5.14: Reconocimiento del patrón dentro de la imagen filtrada con un filtro de vecindad $N = M = 5$

Cuando el medio contiene una distribución aleatoria de dispersores dentro de la celda de resolución. Estos elementos dispersores surgen de las irregularidades y estructuras microscópicas que son más pequeñas que la celda de resolución.

El speckle se puede considerar como ruido, sin embargo, también se puede considerar que aporta información sobre la estructura interna del tejido.

Existen varias aproximaciones para describir el *speckle*. Varios investigadores de imágenes de ultrasonido, usan un análisis estocástico para describir las características del *Speckle*. Debido a que es formado por la suma de ecos de señales, generadas aleatoriamente en los centros de difusión, cada uno proviendo una cantidad aleatoria de energía. Esta demostrado que la estadística del brillo en una imagen de ultrasonido depende mayormente del numero de celdas de resolución y en la distribución espacial de los puntos de difusión.

Tres modelos pueden ser identificados (ver [5]). El primero es para cuando

Figura 5.15: Imagen filtrada con el filtro de vecindad con valores altos de $N = 20$ y $M = 20$

hay un presencia grande de puntos de difusión, modelado por la distribución de *Rayleigh*. El segundo es para cuando agregamos la componente en fase, entonces el brillo puede ser estimado por la distribución de *Rician*. El tercero es cuando existe un bajo numero de puntos efectivos de difusión, en este casa es modelado por la distribución K.

Tiene, por tanto, características aleatorias por estar formada por la suma de señales procedentes de elementos situados en en posiciones aleatorias. Es posible que algunos de estos elementos presenten cierta periodicidad en su colocación, lo que provoca la aparición de una componente determinista en la señal.

Las reflexiones especulares, más fuertes que los ecos procedentes de la dispersión, también contribuyen con componentes deterministas. Podemos afirmar que la señal recibida está formada por componentes aleatorias y componentes deterministas. Para la recepción se utiliza un detector de envolvente (no coherente), por tanto se pierde la fase de la señal. Si la distribución espacial de los dispersores es completamente aleatoria, la fase es una función aleatoria uniforme, por lo que el hecho de descartar la fase de

Figura 5.16: Reconocimiento del patrón dentro de la imagen filtrada con un filtro de vecindad $N = M = 20$

la señal recibida no supone una pérdida importante de la información. Se analizarán las envolventes.

5.3.4. Comandos

Ruido de tipo gaussiano:

```
>> a_ruido=imnoise(a,'gaussian', media, varianza);
```

Ruido de tipo sal y pimienta (impulsivo):

```
>> a_ruido=imnoise(a,'salt & pepper', densidad);
```

Ruido Multiplicativo con distribución uniforme a la imagen i:

```
>> a_ruido=imnoise(a,'speckle', varianza);
```

5.3.5. Experimentos y resultados

Vamos a comenzar mostrando los experimentos realizados con el primer tipo de ruido que en este caso es el Gaussiano. A continuación, pondremos un ejemplo del cálculo de la aplicación de ruido gaussiano a una imagen poniendo los valores de la media a 0 y de la varianza a 0,05 (ver la figura 5.17). Al aplicar el algoritmo vemos que encuentra el patrón, como se observa en la figura 5.18.

Figura 5.17: Imagen con ruido del tipo gaussiano con media 0 y varianza 0,05

A continuación, pondremos un ejemplo para mostrar que el algoritmo también es robusto frente al ruido gaussiano. Para ello subimos el valor de la varianza a 0,3 (ver la figura 5.19) y obtenemos todavía unos resultados precisos en la búsqueda como muestra la figura 5.20.

Vamos a continuar con el segundo tipo de ruido que en este caso es el de Sal y Pimienta. Para este caso utilizamos un valor de la densidad igual a 0,1 (ver la figura 5.21). El algoritmo encuentra el patrón adecuadamente, como vemos en la figura 5.22.

Figura 5.18: Reconocimiento del patrón dentro de la imagen con ruido gaussiano de media 0 y varianza 0,05

A continuación, y como hemos hecho en transformaciones anteriores examinaremos la robustez del algoritmo testándolo con valores más elevados de ruido. Para ello colocamos la densidad a 0,3. Los resultados pueden verse en las figuras 5.23 y 5.24.

Para terminar con los experimentos de los distintos tipos de ruido veremos el multiplicativo. A continuación, pondremos un ejemplo del cálculo de la aplicación de ruido multiplicativo a una imagen poniendo la varianza con un valor de 0,1, como podemos ver en la figura 5.25. A pesar de aplicar ruido multiplicativo a la imagen, el algoritmo ha encontrado el patrón a buscar, como vemos en la figura 5.26.

A continuación, pondremos un ejemplo de hasta donde es preciso el algoritmo y podemos comprobar que hasta que no le aplicamos a la varianza un valor alto, el algoritmo sigue detectando el patrón. Como un caso particular hemos considerado varianza 3, obteniendo los resultados de las figuras 5.27 y 5.28.

Figura 5.19: Imagen con ruido del tipo gaussiano con media 0 y varianza 0,3

5.4. Compresión y formatos de la imagen

Compresión sin pérdida de calidad: La imagen original se puede recuperar a pesar de poder reducir de un 10 % al 40 % el tamaño de la imagen. Ejemplos: TIFF y PNG.

Compresión con pérdida en la calidad: No se puede recuperar la calidad de la imagen original Ejemplos: GIF (mayor de 256 colores) y JPEG.

TIFF (File Image File Format, .tif): Se utilizan para almacenar imágenes de alta calidad. Es el formato preferido de fotógrafos para crear copias impresas.

PNG (Portable Network Graphics, .png): Método de compresión sin pérdida de calidad, ocupa menos espacio que el formato TIFF.

GIF (Graphics Interchange Format, .gif): Representa imágenes de mejor calidad que el formato JPEG en páginas Web.

Figura 5.20: Reconocimiento del patrón dentro de la imagen con ruido gaussiano de media 0 y varianza 0,3

JPEG (Joint Photographic Experts Group, .jpg): Adecuada relación entre el nivel de calidad y el tamaño que ocupa en soportes digitales.

RELACIÓN ENTRE FORMATOS:

FORMATO	TAMAÑO	COMPRESIÓN
Imagen original	$136{,}5kB$	$1x$
JPEG	$46{,}3kB$	$136{,}5kB/46{,}3kB = 2{,}95x$
TIFF	$138{,}5kB$	$136{,}5kB/138{,}5kB = 0{,}985x$
PNG	$96{,}3kB$	$136{,}5kB/96{,}3kB = 1{,}42x$
GIF	$38{,}2kB$	$136{,}5kB/38{,}2kB = 3{,}6x$

Tabla 5.3: Ejemplo de tasas de compresión para una imagen particular con diferentes tipos de formato

Nosotros nos vamos a centrar en este proyecto en la compresión utilizando el formato jpeg, que es uno de los formatos con pérdida que podrían

Figura 5.21: Imagen con ruido del tipo sal y pimienta con densidad 0,1

dar lugar a una modificación de la imagen lo suficientemente importante como para que el algoritmo de reconocimiento de patrones no funcionase correctamente. Como veremos esto no sucede así, y podremos concluir una vez más que el algoritmo es estable.

5.4.1. Compresión JPEG

JPEG (Joint Photograpic Experts Group) es un método de compresión con perdida para imágenes estandarizado por ISO. El concepto de compresión se define como el proceso de conseguir un volumen de datos inferior al original para representar una determinada información.

Fundamentalmente se distinguen dos tipos de compresiones: las compresiones sin pérdida y las compresiones con perdida. Las primeras permiten recuperar la representación original de la información de forma exacta a partir de la versión comprimida. Las compresiones con pérdida permiten solo una reconstrucción aproximada de la representación original. Como se afirmó anteriormente, JPEG es un método de compresión con pérdida, el cuál dispone de un parámetro de calidad. Obviamente a mayor calidad me-

Figura 5.22: Reconocimiento del patrón dentro de la imagen con ruido sal y pimienta con densidad 0,1

nor compresión.

ESQUEMA DE COMPRESIÓN

Supongamos que se dispone de una imagen en color presentada en el modelo RGB (Red Green Blue), con tres matrices M x N que representan un nivel de intensidad de 0 a 255 de cada una de las componentes rojo, verde y azul por cada pixel. Los pasos necesarios para realizar la compresión JPEG de esta imagen serán los que se muestran en el esquema de la figura 5.29.

Los diferentes pasos se detallan a continuación:

1. Transformación del modelo de representación de una imagen en color

El primer paso para realizar la compresión JPEG de una imagen, es obtener su representación en el formato YUV. Este modelo representa en tres matrices las componentes de luminancia (Y), crominancia azul (U) y crominancia roja (V) de una imagen. Teniendo en cuenta que se ha considerado

Figura 5.23: Imagen con ruido sal y pimienta con un valor de densidad 0,3

que la representación de la imagen será proporcionada en el modelo RGB (Red Blue Green), se muestran a continuación las formulas que permiten realizar la conversión del modelo RGB al modelo YUV y viceversa.

$Y = 0{,}299R + 0{,}587G + 0{,}114B$
$U = -0{,}1687R - 0{,}3313G + 0{,}5B + 128$
$V = 0{,}5R - 0{,}4187G - 0{,}0813B + 128$
$R = Y + 1{,}402(V - 128)$
$G = Y - 0{,}3441(U - 128) - 0{,}7139(V - 128)$
$B = Y + 1{,}770(U - 128)$

Tabla 5.4: Conversión entre los formatos de color RGB y YUV

2. Reducción de la resolución de la crominancia

Habitualmente se reduce en un factor de 2, verticalmente y horizontalmente, las componentes de crominancia U y V. Realizando esto, se obtendrían dos nuevas matrices U' y V' cuatro veces mas pequeñas. Un posible

Figura 5.24: Reconocimiento del patrón dentro de la imagen con ruido sal y pimienta de densidad 0,3

método para calcular estas matrices sería realizar la descomposición en bloques de 4 pixels. Posteriormente realizar para cada bloque la media entre sus pixels seguido de un redondeo. Finalmente asignar estos valores a las dos nuevas matrices de resolución 4 veces menor.

Procesado individual de las tres matrices

A partir de este punto realizamos las siguientes acciones de forma independiente para cada una de las tres matrices Y, U' y V'. Consideraremos estas matrices como la matriz X en el proceso siguiente.

3. Centrado

A la matriz X se le resta el valor 128 para que sus valores queden centrados en 0, obtenemos así, una nueva matriz X_c.

4. División en bloques de $8x8$

La matriz X^c se divide en bloques X_b^c de $8x8$ pixels.

Figura 5.25: Imagen con ruido del multiplicativo de varianza 0,1

Procesado individual de cada bloque

Se realizan las operaciones que se describen en los siguientes puntos para cada bloque X_b^c.

5. DCT-2D (Transformada discreta del coseno 2D)

Se la aplica la transformada discreta del coseno en dos dimensiones a la matriz X_b^c. La formula de la DCT-2D es la siguiente:

$$C_b(u,v) = (\frac{2}{M})(\frac{1}{2}) \cdot (\frac{2}{N})(\frac{1}{2}) \cdot \gamma(u) \cdot \gamma(v) \cdot \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \cos[\frac{\pi u}{2M}(2i+1)] \cos[\frac{\pi v}{2N}(2j+1)] \cdot X_b^c(i,j)$$

dónde:

$$\gamma(n) = \begin{cases} 1/\sqrt{2} & \text{si} \quad n = 0 \\ 1 & \text{si} \quad n \neq 0 \end{cases} \tag{5.2}$$

Figura 5.26: Reconocimiento del patrón dentro de la imagen con ruido multiplicativo de varianza 0,1

Una vez realizado este paso se obtiene un nuevo bloque C_b del mismo tamaño con valores más altos en las proximidades de la esquina superior izquierda y valores más bajos en las proximidades de la esquina inferior derecha.

6. Cuantificación

Este paso será el que determine el factor de compresión y la pérdida de la calidad de la imagen.

7. Zig-Zag Scan

Una vez obtenido el bloque cuantificado, representado ahora mediante números enteros, se recorre el bloque en Zig-Zag y se obtiene un vector.

8. Representación RLE y Huffman

En este punto se realiza la codificación entrópica de los vectores obtenidos en el punto anterior.

Figura 5.27: Imagen con ruido multiplicativo con un valor de varianza 3

5.4.2. Comandos

```
>> imwrite(a,'Auxiliar.jpeg','jpeg','Quality', calidad);
```

5.4.3. Experimentos y resultados

Vamos a comenzar mostrando los experimentos realizados con la compresión JPEG. A continuación, pondremos un ejemplo del cálculo de una compresión con valor 25 de calidad (tasa de compresión del 93 %) como vemos en la figura 5.30 y veremos que la búsqueda ha sido llevada a cabo eficientemente porque aunque comprimida la imagen, el algoritmo ha encontrado el patrón, como vemos en la figura 5.31.

Figura 5.28: Reconocimiento del patrón dentro de la imagen con ruido multiplicativo de varianza 3

5.5. Ecualización de histogramas

La ecualización del histograma de una imagen es una transformación que pretende obtener para una imagen un histograma con una distribución uniforme. Es decir, que exista el mismo número de pixels para cada nivel de gris del histograma de una imagen monocroma.

En la transformación, todos los pixels de un mismo nivel de gris se transformarán a otro nivel de gris, y el histograma se distribuirá en todo el rango disponible separando en lo posible las ocupaciones de cada nivel.

Vamos a desarrollar dicho concepto en forma discreta. Para niveles de gris que toman valores discretos, el cálculo de las funciones de densidad de probabilidad viene dado por:

$$p_r(r_k) = \frac{n_k}{n}$$

con $0 \leq r_k \leq 1$ y $k = 0, 1, 2, ..., L-1$, siendo L el número de niveles de gris,

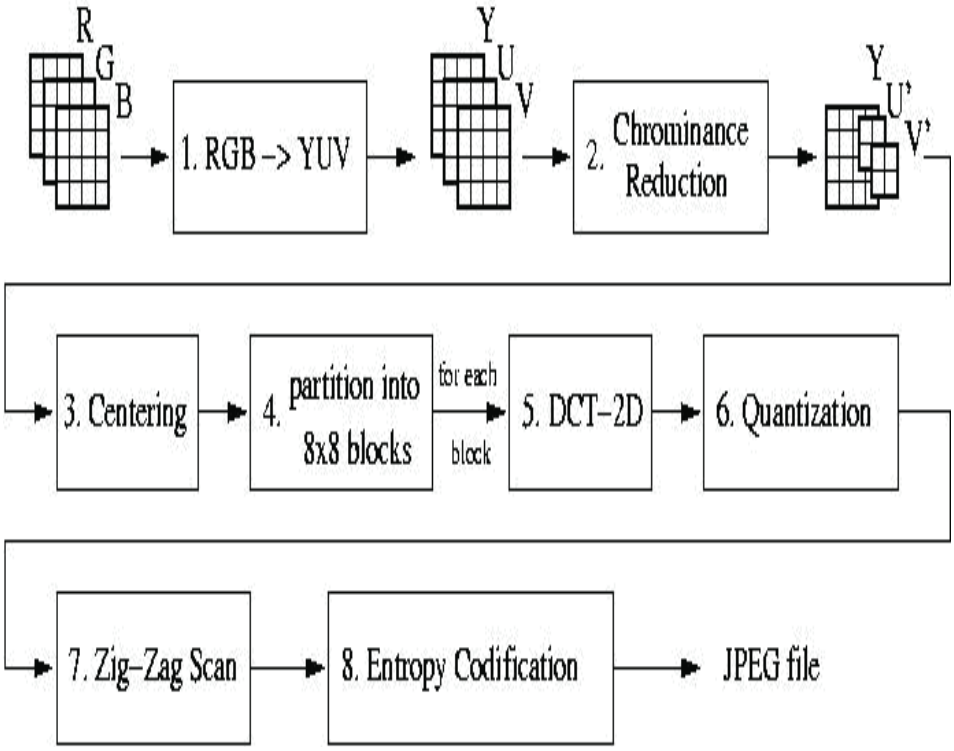


Figura 5.29: Esquema de la compresión JPEG

$p_r(r_k)$ es la probabilidad del k-ésimo nivel de gris en la imagen, y n es el número total de puntos.

Se llama histograma al diagrama de $p_r(r_k)$ frente a r_k. Las técnicas usadas para obtener histogramas uniformes son algoritmos de ecualización. La forma discreta de la ecuación es:

$$s_k = T(r_k) = \sum_{j=0}^{k} \frac{n_j}{n} = \sum_{j=0}^{k} p_r(r_j)$$

con $0 \leq r_k \leq 1$ y $k = 0, 1, 2, ..., L-1$.

Ejemplo de uso de la ecuación:
Supongamos que tenemos una imagen de $64x64$ (4096 pixels) con 8 ni-

Figura 5.30: Imagen comprimida con jpeg con calidad de 25

veles de gris distribuidos según la siguiente tabla:

r_k	n_k	$p_k(r_k) = \frac{n_k}{n}$
$r_0 = 0$	790	0,19
$r_1 = 1/7$	1023	0,25
$r_2 = 2/7$	850	0,21
$r_3 = 3/7$	656	0,16
$r_4 = 4/7$	329	0,08
$r_5 = 5/7$	245	0,06
$r_6 = 6/7$	122	0,03
$r_7 = 1$	81	0,02

Tabla 5.5: Ejemplo de distribución de los píxeles de una imagen en 8 niveles de gris

mediante la funcion de transformación se obtiene:

$$s_0 = T(r_0) = \sum_{j=0}^{0} p_r(r_j) = 0{,}19$$

Figura 5.31: Reconocimiento del patrón dentro de la imagen comprimida con jpeg con calidad 25

$$s_1 = T(r_1) = \sum_{j=0}^{1} p_r(r_j) = 0{,}19 + 0{,}25 = 0{,}44$$

de forma similar se obtiene:

$s_2 = 0{,}65$ $s_3 = 0{,}81$ $s_4 = 0{,}89$ $s_5 = 0{,}95$ $s_6 = 0{,}98$ $s_7 = 1{,}00$

como solo están permitidos 8 valores igualmente espaciados, cada nivel de gris obtenido se debe aproximar a su valor válido más cercano, de manera que nos queda que:

$s_0 = 1/7$, $s_1 = 3/7$, $s_2 = 5/7$, $s_3 = 6/7$, $s_4 = 6/7$, $s_5 = 1$, $s_6 = 1$, $s_7 = 1$.

5.5.1. Comandos

```
>> a=equalcolor(a);
```

5.5.2. Experimentos y resultados

Vamos a mostrar un experimento realizado con la ecualización. En la figura 5.32 vemos la foto de un reloj que carece de luminosidad. En la figura 5.33 el patrón que queremos encontrar dentro de la imagen. En la figura 5.34 mostramos la imagen ecualizada del reloj y en la figura 5.35 la búsqueda del patrón.

Figura 5.32: Imagen original

Figura 5.33: Patrón de búsqueda

Figura 5.34: Imagen ecualizada

Figura 5.35: Reconocimiento del patrón dentro de la imagen ecualizada

Capítulo 6

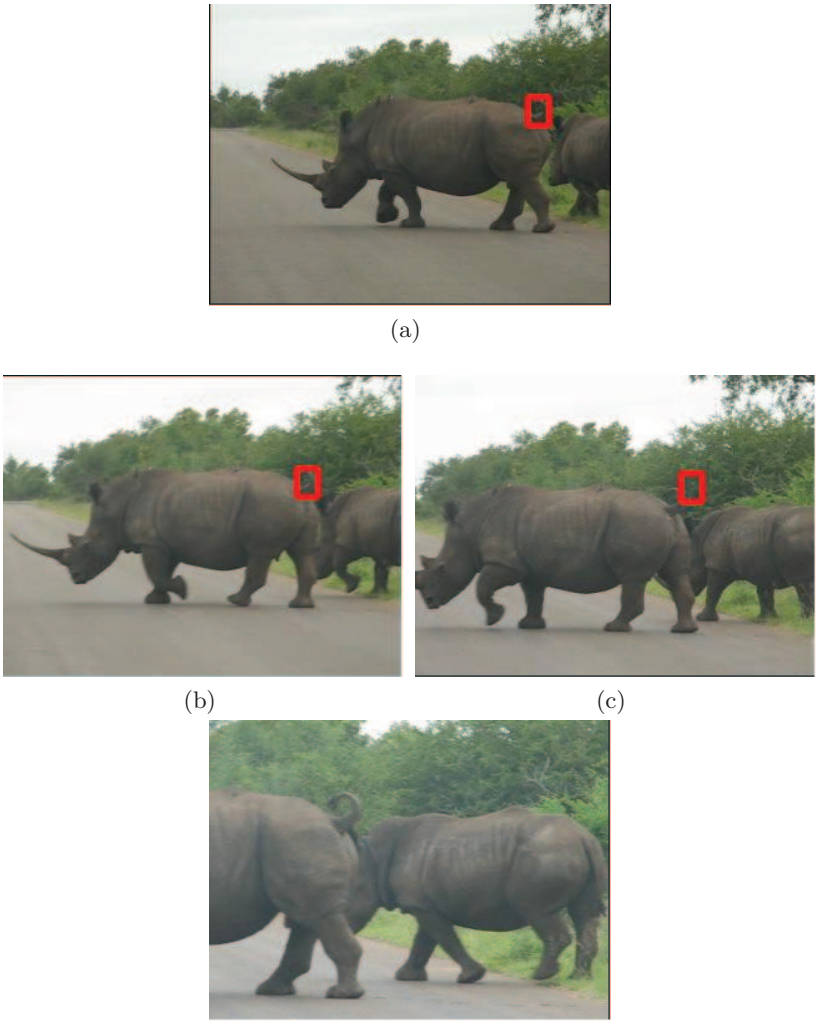
Definición del Algoritmo en 3D. Aplicación a las secuencias de vídeo

En este capítulo vamos a presentar un algoritmo de reconocimiento de patrones para señales en tres dimensiones. El algoritmo está basado principalmente en el filtro dado en las fórmulas (2.1)-(2.2), aunque utilizamos unas condiciones complementarias que presentamos a continuación. Vamos a explicar el algoritmo paso a paso con la ayuda de la Figuras 6.1 y 6.2.

Supongamos que tenemos discretizada la secuencia de vídeo de manera que estamos trabajando en un espacio finito dimensional. Igualmente tenemos también un patrón a buscar dentro de la señal, el cual es una imagen discretizada, llamemosla V, como muestra la Figura 6.2.

El primer paso del algoritmo será ir recorriendo todos los fotogramas del vídeo 6.1 y para cada uno de ellos aplicar el filtro presentado en el capítulo de señales dos dimensionales. Para ello, según se explicó en dicho capítulo seguimos los siguientes pasos. Normalizamos el patrón V haciéndolo así unitario. Para ello utilizamos la fórmula:

$$\bar{V}_{ij} = \frac{V_{ij}}{\sqrt{\sum_i \sum_j V_{ij}^2}}. \tag{6.1}$$



(a) (b) (c) (d)

Figura 6.1: Imágenes de unos rinocerontes donde percibimos la sensación de movimiento

Podemos suponer de esta manera que V tiene norma 1. El segundo paso será ir tomando muestras del fotograma del mismo tamaño que V. LLamemos U a una de estas muestras. El camino a seguir consistirá en normalizar U para hacerlo unitario y pasar entonces el filtro con entradas las matrices U y V, las cuales previamente las hemos dispuesto en forma de vector. Este proceso deberá realizarse con cada una de las posibles muestras U con-

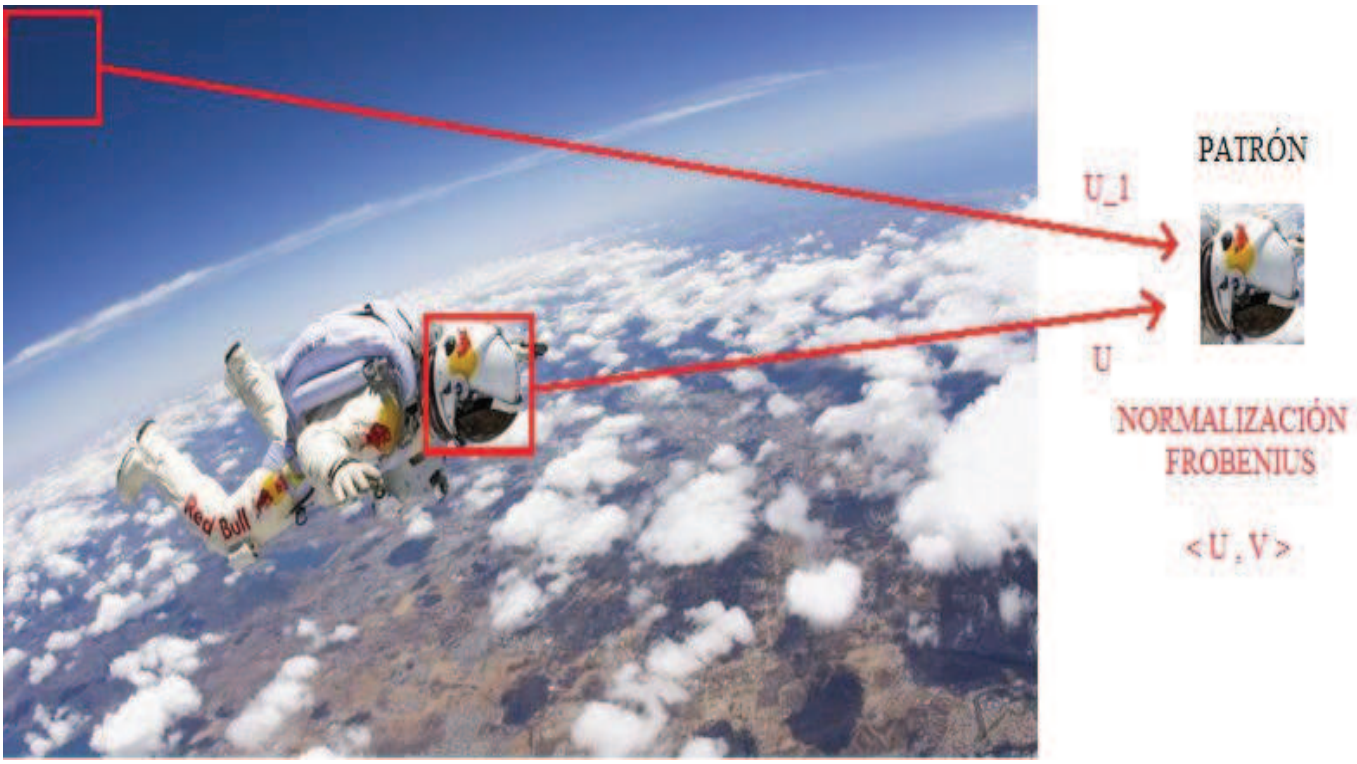


Figura 6.2: Explicación gráfica del filto basado en la desigualdad de Cauchy-Schwartz

tenidas en dicho fotograma (ver Figura 6.2), comparando a cada paso los valores del filtro, y quedándonos con las posiciones donde el valor del filtro es mayor que un sesgo y que además cumplen el requisito de continuidad temporal y la condición de que la media de los valores de los píxeles en la muestra asociada dista poco de la media de los valores de los píxeles del template. La continuidad temporal significa que el patrón en un fotograma debe estar cerca del lugar donde estaba en el fotograma anterior. En el caso del primer fotograma esta condición es trivial porque se conoce la posición inicial del template. Entre todas las posiciones factibles elegimos la que da un mayor valor del filtro. Esa será la posición donde el algoritmo detecta el patrón dentro de la señal. En el caso del ejemplo de la Figura 6.2 el valor del filtro será exactamente 1 para la muestra contenida en la matriz U_2.

Esquemáticamente podemos expresar los diferentes pasos del algoritmo

de la siguiente manera:

Algoritmo 3D

- Obtener las versiones discretizadas de la secuencia de video y del template
- Normalizar el template V
- Calcular la media de sus píxeles
- Para cada uno de los fotogramas del vídeo hacer:
  - Para cada una de las posibles muestras U del tamaño del template contenidas en la imagen hacer:
    - Normalizar U
    - Calcular el filtro $< U, V >$
    - Guardar las posiciones que dan un filtro mayor que un sesgo y cumplen el requisito de continuidad y la condición de la media
    - Calcular la media de los píxeles de U
    - De entre las posiciones seleccionadas elegir la que da un mayor valor del filtro
    - La variable *posicion* contiene la localización del template

6.1. Aplicación a los patrones de vídeo

Hemos desarrollado una interfaz gráfica que permite al usuario ejecutar los programas realizados en lenguaje Matlab que llevan a cabo el reconocimiento de patrones descrito de una manera sencilla.

En la Figura 6.3 se muestra la interfaz gráfica creada. En ella vemos que en la parte de la izquierda se encuentran las opciones para cargar un vídeo y un patrón a buscar.

Una vez cargados el vídeo y el patrón elegidos se ejecuta el programa pinchando en el botón *Aplicar* en la parte inferior derecha de la interfaz.

Vamos a llevar a cabo una búsqueda en concreto. Para ello cargamos el vídeo $'iniesta.avi'$ que contiene un vídeo real de la jugada del gol de la final

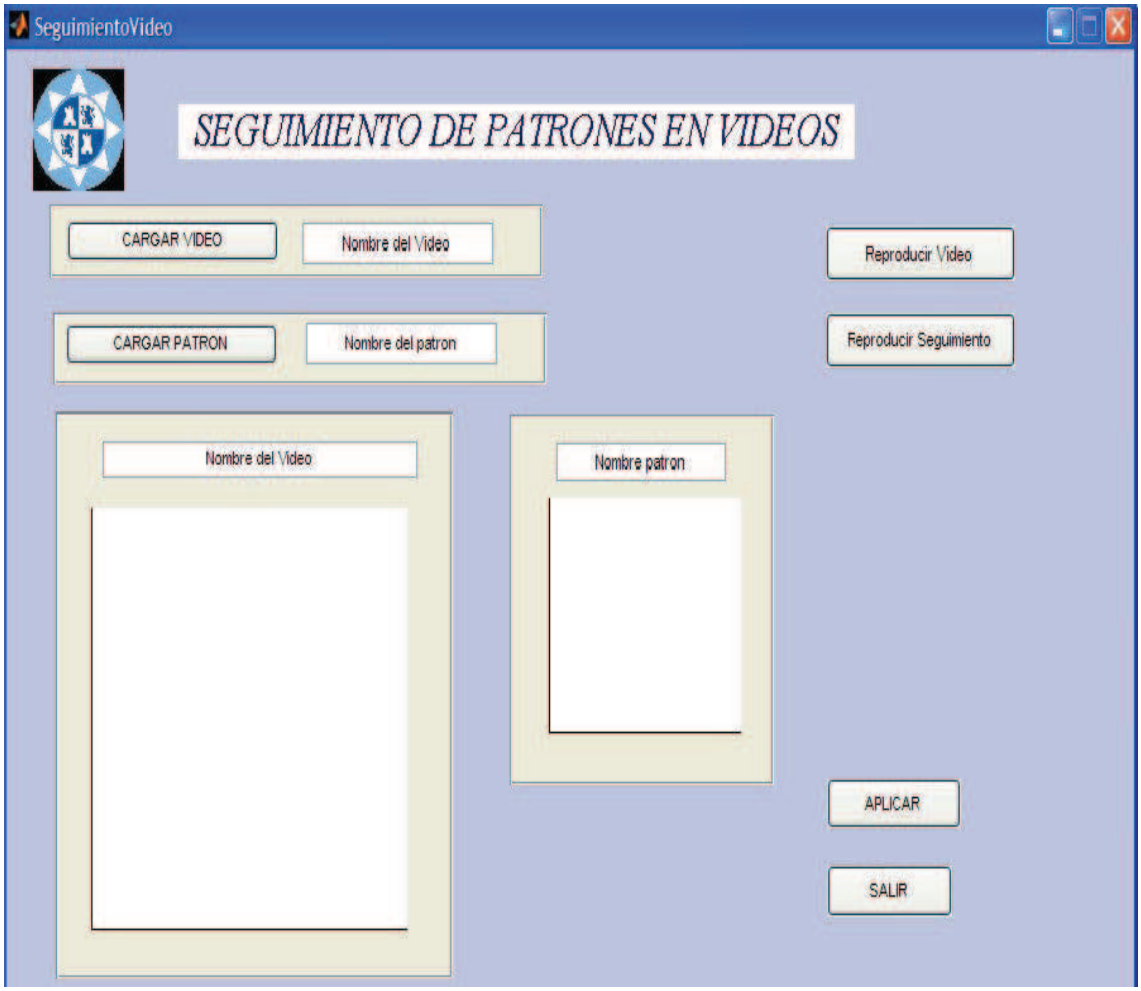


Figura 6.3: Interfaz para reconocimiento de patrones en vídeos

del Mundial de fútbol del año 2010 disputada entre España y Holanda (ver Figura 6.4). En primer lugar cargaremos el vídeo *'iniesta.avi'* incorporando la ruta para que el programa lo localice y el vídeo sea cargado sin problemas.

El segundo paso consistirá en leer el template de búsqueda, en nuestro caso la muestra de la imagen 'template-iniesta.png' (ver Figura 6.4) añadiéndole la ruta del directorio donde tenemos la imagen.

A continuación podemos ***reproducir el vídeo*** de la jugada del gol si lo

Figura 6.4: Cargamos el vídeo *'iniesta.avi'* y su respectivo template

deseamos. Una vez seleccionados el vídeo y el template le damos al botón ***Aplicar***. Al terminar el programa podemos ***reproducir el seguimiento*** hecho del template, es decir, podemos seguir el balón, que en este caso sería el patrón durante toda la jugada hasta que se produzca ***oclusión***. Por ejemplo, el patrón es reconocido durante todo el seguimiento hasta que el jugador Iniesta golpea el balón y la detección del patrón no se realiza con éxito (ver Figura 6.7.

El resultado de aplicar este algoritmo sobre la imagen es reconocer el

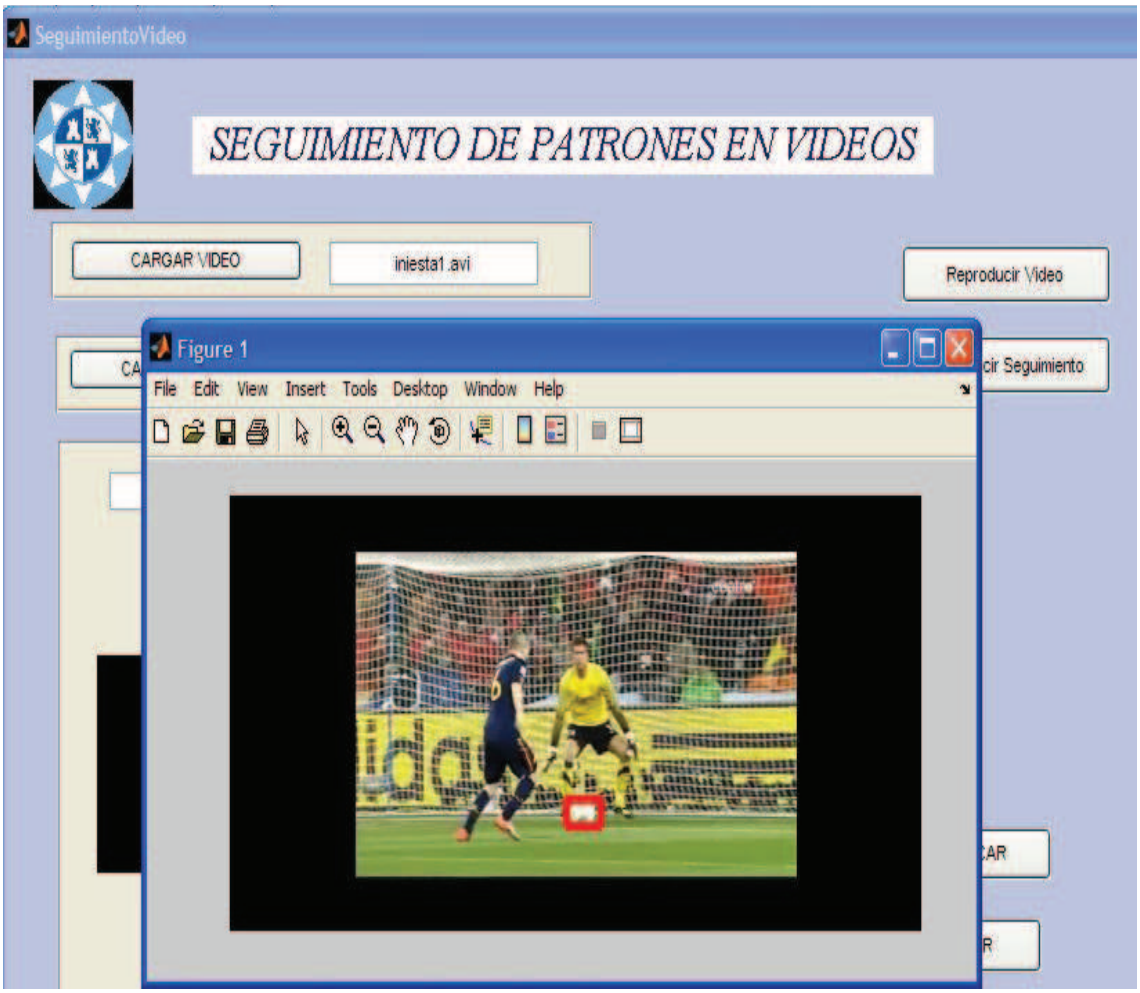
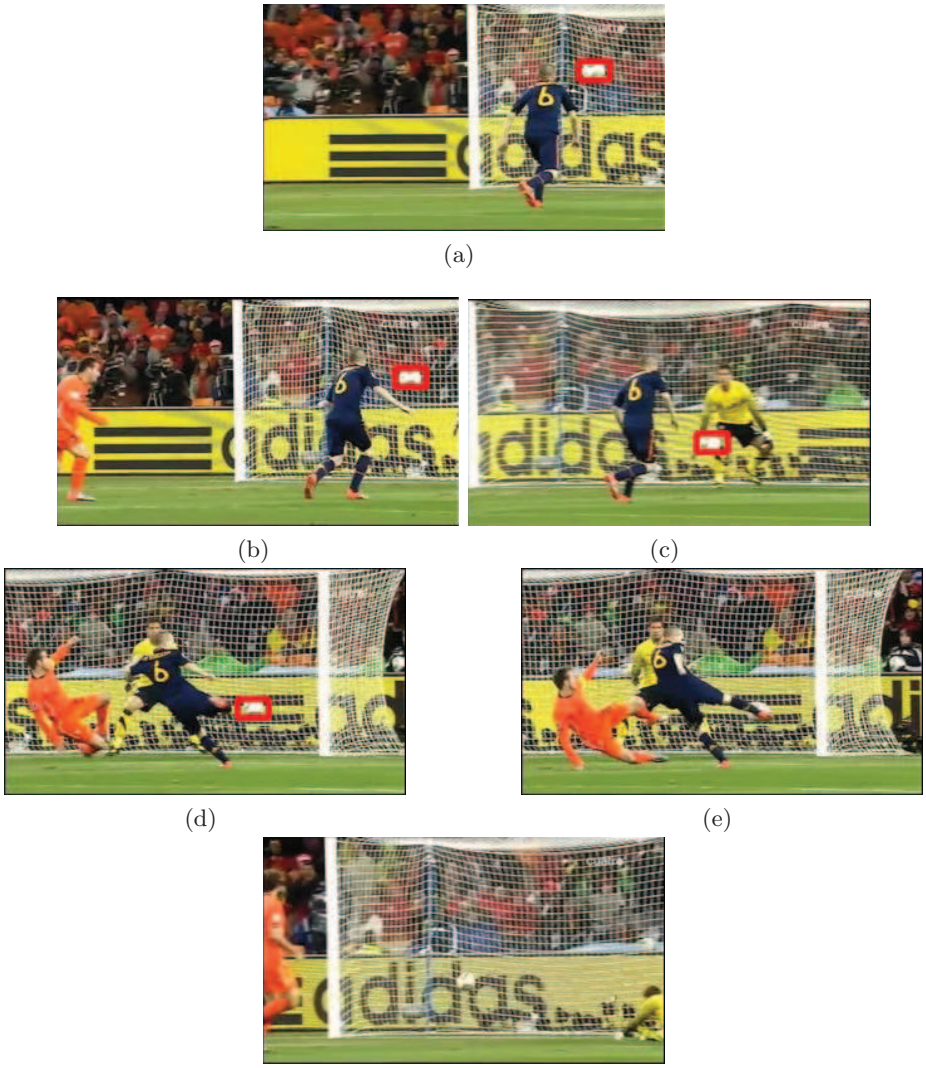


Figura 6.5: Reconocimiento del patrón de búsqueda del vídeo *'template – iniesta.png'*

template ó patrón de búsqueda dentro de cada fotograma mediante un cuadrado de color rojo, como se puede ver en las Figuras 6.2, 6.5 y 6.7.

Figura 6.6: Resultado de aplicar el algoritmo con la detección del patrón en un caso cercano a la oclusión



(a)

(b)

(c)

(d)

(e)

(f)

Figura 6.7: Imagen del funcionamiento de la búsqueda del patrón hasta el momento en el que se produce la oclusión

Capítulo 7

Desarrollo de los programas en Matlab

7.1. Código fuente del algoritmo en 1D para procesado de señales de audio

7.1.1. Programa en 1D sin interfaz gráfica

```
function patrones_1D (v,template)

% patrones_1D (v,template);
% Variables de entrada:
% v vector sobre el que buscar un patrón
% template patrón de búsqueda


% cálculo de las longitudes de los vectores

n=length(v); m=length(template);

% normalizamos el template

template_n=template/norm(template,'fro')

% recorremos el vector v y cada m coordenadas hacemos el producto
escalar euclideo con el template tras haber previamente normalizado
ambos vectores
```

```
for i=1:n-m+1

% vamos tomando muestras en el vector v del mismo tamaño que el
template


    muestra=v(i:i+m-1);


   % normalizamos la muestra tomada


    muestra=muestra/norm(muestra,'fro')


   % calculamos el producto escalar en norma 2 de el template normalizado
con la muestra normalizada

    p(i)=sum(muestra.*template_n);
end

% calculamos el máximo valor del filtro

[maxVal,x]=max(p)
```

7.2. Código fuente del algoritmo en 2D para procesado de imágenes digitales

7.2.1. Funciones previamente definidas en Matlab

Funcion EQUALCOLOR

```
function b=equalcolor(a)

% b=equalcolor(a);
% Objetivos:
% leer una imagen y ecualizarla. Si la imagen es de color, ecualiza
cada banda por separado. Ecualizar por bandas separadas no siempre
funciona bien.
% Variables de entrada: a imagen a ecualizar
% Variables de salida: b imagen ecualizada
```

```
a=uint8(a); [n,m,l]=size(a);

% ecualización de cada banda


b=zeros(n,m,l,'uint8');

for i=1:l

    b(:,:,i)=histeq(a(:,:,i));

end
```

Experimento: Crear funcion Patrones_2D

El primer paso para realizar el código en Matlab de los programas es crear una función llamada Patrones_2D que realiza la búsqueda de un template (matriz pequeña) dentro de una matriz dada. Este programa no esta conectado mediante una interfaz gráfica sino que se ejecuta directamente.

```
function patrones_2D(a,template)

% patrones_2D(a,template);
% Variables de entrada:
% a matriz sobre la que buscar un patrón
% template patrón de búsqueda

% cálculo de las dimensiones de las matrices

[ng,mg]=size(a); [np,mp]=size(template);

% normalizamos el template

template_n=template/norm(template,'fro');

% recorremos la matriz a y para cada submatriz de tamaño npxmp con
filas y columnas correlativas hacemos el producto escalar en la
norma frobenius con el template tras haber previamente normalizado
ambas matrices
```

```
for i=1:ng-np+1
    for j=1:mg-mp+1

% vamos tomando muestras en la matriz a del mismo tamaño que el
template

    muestra=a(i:i+np-1,j:j+mp-1);

% normalizamos la muestra tomada

    muestra=muestra/norm(muestra,'fro');

% calculamos el producto escalar en norma frobenius del template
normalizado con la muestra normalizada

    p(i,j)=sum(sum(muestra.*template_n));
    end
end

% calculamos el máximo valor del filtro

[maxVec,x]=max(p); [maxVal,y]=max(maxVec);

[x(y),y] maxVal
```

7.2.2. Programa en 2D sin interfaz gráfica

```
function [coor,maxVal]=impatrones_2D(im,name_template)

% [coor, maxVal] = impatrones_2D(im, name_template);
% Variables de salida:
% coor vector con las coordenadas [x, y] del punto donde se
% alcanza el mayor valor del filtro
% maxVal valor máximo alcanzado por el filtro
% Variables de entrada:
% im nombre de la imagen sobre la que buscar un patrón
% name_template imagen que constituye el patrón de búsqueda

close all

% lectura de las imágenes
```

```
a=imread(im); a=double(a); template=imread(name_template);
template=double(template);

% cálculo de las dimensiones de las imágenes

[ng,mg,lg]=size(a); [np,mp,lp]=size(template);


if (lg~=lp)
    fprint('La imagen y el patrón deben ser ambas imágenes en escala ...
    de grises o imágenes en color\n');
    return;
end

% inicialización de las matrices auxiliares q y p

q=zeros(ng-np+1,mg-mp+1); p=zeros(ng-np+1,mg-mp+1,lg);

% bucle para recorrer las bandas de color

for k=1:lg

% normalizamos el template

 template_n(1:np,1:mp,k)=template(1:np,1:mp,k)/(norm(template(1:np,1:mp,k),'fro')...
 +10^(-10));

% recorremos la matriz a y para cada submatriz de tamaño npxmp con
filas y columnas correlativas hacemos el producto escalar en la
norma frobenius con el template tras haber previamente normalizado
ambas matrices

for i=1:ng-np+1
    for j=1:mg-mp+1

% vamos tomando muestras en la matriz a del mismo tamaño que el
template

    muestra(1:np,1:mp)=a(i:i+np-1,j:j+mp-1,k);


    muestra=muestra/(norm(muestra,'fro')+10^(-10));
```

```
% calculamos el producto escalar en norma frobenius del template
normalizado con la muestra normalizada

    p(i,j,k)=sum(sum(muestra.*template_n(:,:,k)));
    q(i,j)=q(i,j)+p(i,j,k);

        end
    end
end

% calculamos el mayor valor del filtro

[maxVec,x]=max(q); [maxVal,y]=max(maxVec);

coor(1)=x(y); coor(2)=y;

% mostramos las diferentes figuras

clim_inf=min(min(a)); clim_sup=max(max(a));
figure('Tag','Original','Name',[blanks(6),'IMAGEN ORIGINAL'],...
'Position',[6 25 1270 710])
hy=min([1,ng/710]); hx=min([1,mg/1270]); axes('Position',[0.5-hx/2
0.5-hy/2 hx hy]) if (lg==1)
imagesc(a,[clim_inf,clim_sup]); colormap gray
    else
        a=uint8(a);
        imshow(a)
        end axis off
pause
figure('Tag','Template','Name',[blanks(6),'Template'],...
'Position',[6 25 1270 710])
ax=hx*mp/mg; ay=hy*np/ng; axes('Position',[0.5-ax/2 0.5-ay/2 ax
ay]) if(lg==1)
imagesc(template,[min(min(template)),max(max(template))]);
colormap gray;
    else
        template=uint8(template);
        imshow(template);
        end axis off
pause
figure('Tag','Original','Name',[blanks(6),'LOCALIZACIÓN DEL
PATRÓN'],...
```

```
'Position',[6 25 1270 710])
hy=min([1,ng/710]); hx=min([1,mg/1270]); axes('Position',[0.5-hx/2
0.5-hy/2 hx hy]) if (lg==1)
imagesc(a,[clim_inf,clim_sup]); colormap gray
    else
        a=uint8(a);
        imshow(a);
        end
hold on rect_posicion=[coor(2) coor(1)  mp np];
rectangle('Position',rect_posicion
'LineWidth',3,...
'EdgeColor',[1 0 0],...
'LineStyle','-');
axis off
```

7.3. Código fuente del algoritmo en 3D

7.3.1. Programa para la interfaz gráfica en 3D

```
function varargout = SeguimientoVideo(varargin);

   % Código de inicialización

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @SeguimientoVideo_OpeningFcn, ...
                   'gui_OutputFcn',  @SeguimientoVideo_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
function varargout = SeguimientoVideo(varargin);
```

```
    % Ejecuta justo antes y SeguimientoVideo se hace visible

function SeguimientoVideo_OpeningFcn(hObject, eventdata, handles, varargin);

    % Elija el comando de salida por defecto de línea de SeguimientoVideo


handles.output = hObject;

    % Actualizar la estructura de datos

guidata(hObject, handles);

    % Colocar imagen de fondo

background=imread('escudo_upct.png');
axes(handles.axes3);
axis off;
imshow(background);

    % Las salidas de esta función devuelven a la línea de comandos


function varargout = SeguimientoVideo_OutputFcn(hObject, eventdata, handles);

    % Obtenga el comando predeterminado de salida de la estructura
de datos

varargout{1} = handles.output;

function varargout = SeguimientoVideo_OutputFcn(hObject, eventdata, handles);

    % Realiza la llamada de la función

function edit4_Callback(hObject, eventdata, handles);

    % Ejecuta para crear el objeto cuando ajusta todas las propiedades


function edit4_CreateFcn(hObject, eventdata, handles);

    % Ejecuta para crear el objeto cuando ajusta todas las propiedades
```

```
function edit4_CreateFcn(hObject, eventdata, handles);

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end

function edit2_Callback(hObject, eventdata, handles);

   % Ejecuta para crear el objeto cuando ajusta todas las propiedades


function edit2_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end

function edit1_Callback(hObject, eventdata, handles)

   % Ejecuta para crear el objeto cuando ajusta todas las propiedades


function edit1_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end

   % Ejecuta el botón presionando pushbutton1


function pushbutton1_Callback(hObject, eventdata, handles)

addpath(genpath('.\Videos'));

[filename,pathname,filterindex]=uigetfile( ...
    { '*.avi', 'Vídeo en el que buscar'},'Seleccione un vídeo');
```

```
if filterindex == 1
    set(handles.edit1,'String',filename);
    set(handles.edit3,'String',filename);

   % Leemos el vídeo


    video1=aviread(filename);
    fotograma1=video1(1).cdata(:,:,:);
    axes(handles.axes1);
    axis off;
    imshow(fotograma1);
end

rmpath(genpath('.\Videos'));

   % Ejecuta el botón presionando pushbutton2

function pushbutton2_Callback(hObject, eventdata, handles)

addpath(genpath('.\Patrones'));

[filename,pathname,filterindex]=uigetfile( ...
{ '*.jpg', 'Imágenes con formato jpg (*.jpg/jpeg)'; ...
   '*.pgm', 'Imágenes con formato pgm (*.pgm)';...
'*.pbm', 'Imágenes con formato pbm (*.pbm)';...
'*.png', 'Imágenes con formato png (*.png)';...
'*', 'Cualquier fichero'},...
'Elige una imagen');

if filterindex ==1 | filterindex ==2 | filterindex ==3 |
filterindex ==4 | filterindex ==5
set(handles.edit2,'String',filename);
set(handles.edit4,'String',filename);
patron=imread(filename);
axes(handles.axes2);
axis off;
imshow(patron);
end

rmpath(genpath('.\Patrones'));
```

```
   % Ejecuta el botón presionando pushbutton3

function pushbutton3_Callback(hObject, eventdata, handles)

   % Obtenemos el archivo a reproducir

video=get(handles.edit1,'String');
if isempty(video)==1
uiwait(msgbox('Tiene que seleccionar un video','Mensaje de error','error','modal'));
return;
end

addpath(genpath('.\Videos'));

play_avi(video);

rmpath(genpath('.\Videos'));

   % Ejecuta el botón presionando pushbutton4

function pushbutton4_Callback(hObject, eventdata, handles)

   % Obtenemos el archivo a reproducir

video_patron=get(handles.edit1,'String');
lv=length(video_patron);
video_patron=[video_patron(1:lv-4),'_patron',video_patron(lv-3:lv)];

if isempty(video_patron)==1
uiwait(msgbox('Tiene que seleccionar un video','Mensaje de error','error','modal'));
return;
end

addpath(genpath('.\Búsqueda'));

play_avi(video_patron);

rmpath(genpath('.\Búsqueda'));

   % Ejecuta el botón presionando pushbutton5

function pushbutton5_Callback(hObject, eventdata, handles)
```

```
   % Lee el video y el patrón a buscar

addpath(genpath('.\Búsqueda'));
addpath(genpath('.\Videos'))
addpath(genpath('.\Patrones'))

video_name=get(handles.edit1,'String');
name_template=get(handles.edit2,'String');
info_fichero=aviinfo(video_name);
fps=info_fichero.FramesPerSecond;
nf=info_fichero.NumFrames;
video=aviread(video_name);
template=imread(name_template);

   % Llamamos al algoritmo en 2D para cada fotograma e inicialización
de variables

oclu=1;
coor=[1,1];

   % Recorremos los distintos fotogramas

for i=1:nf
    im=video(i).cdata(:,:,:);
    [coor,maxVal,im1,oclu]=impatron_2D(im,template,coor,oclu);
    video_patron(i).cdata=im1;
    video_patron(i).colormap=video(i).colormap;
end

lv=length(video_name);
nuevo_video=['./Búsqueda/',video_name(1:lv-4),'_patron',
video_name(lv-3:lv)];
movie2avi(video_patron,nuevo_video,'compression',
'None','quality',0,'fps',fps);

play_avi(nuevo_video);

rmpath(genpath('.\Búsqueda'));
rmpath(genpath('.\Videos'));
rmpath(genpath('.\Patrones'));
```

7.3.2. Programa para la visualización del vídeo

```
function play_avi(nombre_video);

% Esta función muestra un vídeo : play_avi(nombre_video)
% Variables de entrada : nombre_video
% Lee el vídeo

video=aviread(nombre_video);
info_avi=aviinfo(nombre_video);

% obtiene el tamaño

height=info_avi.Height;
width=info_avi.Width;
height_figure=6/5*height;

width_figure=6/5*width;
rect1=[200,200,width_figure,height_figure,];
if width>=height
    rect2_h=height/width*0.8;
    rect2=[0.1 0.1 0.8 rect2_h];
else
    rect2_w=width/height*0.8;
    rect2=[0.1 0.1 rect2_w 0.8];
end

figure('position', rect1);
h=axes('position',rect2);
axis off;
movie(h,video,1,info_avi.FramesPerSecond);
```

7.3.3. Programa de reconocimiento de patrones de vídeo

```
function [coor,maxVal,im1,oclu]=impatron_2D(im,template,coor,oclu);

% [coor, maxVal, im1, oclu] = impatron_2D(im, template, coor, oclu)
% Variables de salida:
% coor: vector con las coordenadas [x, y] del punto donde se alcanza
el mayor valor del filtro
% maxVal: valor máximo alcanzado por el filtro
```

```
% im1: video con el patrón marcado
% oclu: indica si hay oclusión en el fotograma analizado
% Variables de entrada:
% im: nombre de la imagen sobre la que buscar un patrón
% template: imagen que constituye el patrón de búsqueda
% coor: coordenadas donde estaba el patrón en el fotograma anterior
% oclu: indica si ha habido oclusión en el fotograma anterior

% Inicialización de variables

maxVal=-1;

% Variación permitida en el valor medio de los píxeles

ndb=40;

% Número de píxeles que definen una vecindad

pp=10;

% Lectura de las imágenes

a=im;

% Cálculo de las dimensiones de las imágenes

[ng,mg,lg]=size(a);
[np,mp,lp]=size(template);
if (lg~=lp)
    display('La imagen y el patrón deben ser ambas imágenes en
    escala de grises o imágenes en color\n');
    return;
end

% Transformamos las imágenes a double

a=double(a); template=double(template);

% Inicialización de las matrices que guardan los productos escalares


q=zeros(ng-np+1,mg-mp+1);
p=zeros(ng-np+1,mg-mp+1,lg);
valor_medio=zeros(ng-np+1,mg-mp+1,lg);
```

```
% CASO EN QUE HAY OCLUSIÓN EN EL FOTOGRAMA ANTERIOR

if oclu==1
    display('Oclusión en el Fotograma Anterior o es el Primer Fotograma')
    for k=1:lg

% Calculamos el valor medio de los pixeles del template

valor_medio_template(k)=sum(sum(template(:,:,k)))/np/mp;

% Normalizamos el template

template\_n(1:np,1:mp,k)=template(1:np,1:mp,k)/

(norm(template(1:np,1:mp,k),'fro')+10^(-10));

% recorremos la matriz a y para cada submatriz de tamaño npxmp con
filas y columnas correlativas hacemos el producto escalar en la
norma frobenius con el template tras haber previamente normalizado
ambas matrices

        for i=1:ng-np+1
            for j=1:mg-mp+1

% Vamos tomando muestras en la matriz a del mismo tamaño que el
template

muestra(1:np,1:mp)=a(i:i+np-1,j:j+mp-1,k);

% Calculamos el valor medio de la muestra

valor\_medio(i,j,k)=sum(sum((muestra)))/np/mp;

% Normalizamos la muestraN

muestra=muestra/(norm(muestra,'fro')+10^(-10));

% calculamos el producto escalar en norma frobenius del template
normalizado con la muestra normalizada
```

```
p(i,j,k)=sum(sum(muestra.*template_n(:,:,k)));
 q(i,j)=q(i,j)+p(i,j,k);
            end
        end
    end

% Nos quedamos con las muestras cuyo valor medio para cada una de
las bandas diste menos de ndb db del valor medio del template para
esa banda

[pos1x,pos1y]=find(abs(valor_medio(:,:,1)-

valor_medio_template(1))<ndb);

[pos2x,pos2y]=find(abs(valor_medio(:,:,2)-

valor_medio_template(2))<ndb);

[pos3x,pos3y]=find(abs(valor_medio(:,:,3)-

valor_medio_template(3))<ndb);

% Nos quedamos también con las posiciones dónde se da un valor del
filtro mayor que un sesgo

sesgo=2.998;
[pos4x,pos4y]=find(q>sesgo);

% Matrices que contienen los índices factibles

pos1=[pos1x pos1y];
pos2=[pos2x  pos2y];
pos3=[pos3x pos3y];
pos4=[pos4x pos4y];

% Nos quedamos con las posiciones que aparecen en todas las listas


pos=[];

for i=1:size(pos4,1)
```

```
    for j=1:size(pos3,1)
        if pos4(i,:)-pos3(j,:)==[0,0]
            for k=1:size(pos2,1)
                if pos4(i,:)-pos2(k,:)==[0,0]
                    for l=1:size(pos1,1)
                        if  pos4(i,:)-pos1(l,:)==[0,0]
                            pos=[pos;[pos4(i,1),pos4(i,2)]];
                        end
                    end
                end
            end
        end
    end
end

% Si la lista de posiciones posibles no es vacía, suponemos que
hemos encontrado el patrón, guardamos las coordenadas de la posición
donde lo hemos encontrado dibujamos un rectángulo en la imagen señalando
el patrón

    if ~isempty(pos)

% Suponemos que hemos encontrado el patrón y por tanto la ponemos
oclusión a 0

    oclu=0;

% Buscamos de las posibles posiciones la que da mayor valor de filtro


coor(1)=pos(1,1); coor(2)=pos(1,2);
maxVal=q(pos(1,1),pos(1,2));

for i=2:size(pos,1)
    aux=q(pos(i,1),pos(i,2));
    if aux>maxVal
        coor(1)=pos(i,1); coor(2)=pos(i,2);
        maxVal=q(pos(i,1),pos(i,2));
    end
end
```

```
% Pintamos el rectángulo rojo alrededor del patrón

a(coor(1):coor(1)+np,coor(2)-2:coor(2)+2,1)=255;
a(coor(1):coor(1)+np,coor(2)-2:coor(2)+2,2)=0;
a(coor(1):coor(1)+np,coor(2)-2:coor(2)+2,3)=0;

a(coor(1):coor(1)+np,coor(2)+mp-2:coor(2)+mp+2,1)=255;
a(coor(1):coor(1)+np,coor(2)+mp-2:coor(2)+mp+2,2)=0;
a(coor(1):coor(1)+np,coor(2)+mp-2:coor(2)+mp+2,3)=0;

a(coor(1)-2:coor(1)+2,coor(2):coor(2)+mp,1)=255;
a(coor(1)-2:coor(1)+2,coor(2):coor(2)+mp,2)=0;
a(coor(1)-2:coor(1)+2,coor(2):coor(2)+mp,3)=0;

a(coor(1)+np-2:coor(1)+np+2,coor(2):coor(2)+mp,1)=255;
a(coor(1)+np-2:coor(1)+np+2,coor(2):coor(2)+mp,2)=0;
a(coor(1)+np-2:coor(1)+np+2,coor(2):coor(2)+mp,3)=0;

end

% oclu=0, caso en que conocemos las coordenadas del patrón anteriormente


else
    display('Sin Oclusión en el Fotograma Anterior')
    for k=1:lg

% Calculamos el valor medio de los pixeles del template

valor\_medio\_template(k)=sum(sum(template(:,:,k)))/np/mp;

% Normalizamos el template

template_n(1:np,1:mp,k)=template(1:np,1:mp,k)/

(norm(template(1:np,1:mp,k),'fro')+10^(-10));

% Recorremos un entorno de la posición anterior del patrón en la
matriz a y para cada submatriz de tamaño npxmp con filas y columnas
correlativas hacemos el producto escalar en la norma frobenius con
el template tras haber previamente normalizado ambas matrices
```

```
        for i=coor(1)-pp:coor(1)+pp
            for j=coor(2)-pp:coor(2)+pp

% Podemos tomar la muestra dentro de la imagen

if i+np-1<=ng & j+mp-1<=mg

% Vamos tomando muestras en la matriz a del mismo tamaño que el
template

muestra(1:np,1:mp)=a(i:i+np-1,j:j+mp-1,k);

% Calculamos el valor medio de la muestra

valor\_medio(i,j,k)=sum(sum((muestra)))/np/mp;

% Normalizamos la muestra

muestra=muestra/(norm(muestra,'fro')+10^(-10));

% Calculamos el producto escalar en norma frobenius de el template
normalizado con la muestra normalizada

p(i,j,k)=sum(sum(muestra.*template_n(:,:,k)));
q(i,j)=q(i,j)+p(i,j,k);
    else
        q(i,j)=0;
            end
        end
    end
end

% Nos quedamos con las muestras cuyo valor medio para cada una de
las bandas diste menos de ndb db del valor medio del template para
esa banda

[pos1x,pos1y]=find(abs(valor_medio(coor(1)-pp:coor(1)+pp,coor(2)-

pp:coor(2)+pp,1)-valor_medio_template(1))<ndb);

[pos2x,pos2y]=find(abs(valor_medio(coor(1)-pp:coor(1)+pp,coor(2)-
```

```
pp:coor(2)+pp,2)-valor_medio_template(2))<ndb);


[pos3x,pos3y]=find(abs(valor_medio(coor(1)-pp:coor(1)+pp,coor(2)-

pp:coor(2)+pp,3)-valor_medio_template(3))<ndb);

pos1x=pos1x+coor(1)-pp-1;
pos1y=pos1y+coor(2)-pp-1;
pos2x=pos2x+coor(1)-pp-1;
pos2y=pos2y+coor(2)-pp-1;
pos3x=pos3x+coor(1)-pp-1;
pos3y=pos3y+coor(2)-pp-1;

% Matrices que contienen los índices factibles

    pos1=[pos1x pos1y];
    pos2=[pos2x  pos2y];
    pos3=[pos3x pos3y];

% Nos quedamos con las posiciones que aparecen en todas las listas


pos=[];

for i=1:size(pos1,1)
    for j=1:size(pos2,1)
        if pos1(i,:)-pos2(j,:)==[0,0]
            for k=1:size(pos3,1)
                if pos1(i,:)-pos3(k,:)==[0,0]
                   pos=[pos;[pos1(i,1),pos1(i,2)]];
                end
            end
        end
    end
end

% Nos quedamos con las posiciones que aparecen en todas las listas


    if ~isempty(pos)
```

```
% Entre las posiciones posibles nos quedamos elegimos en la que
el valor del filtro es mayor

coor(1)=pos(1,1); coor(2)=pos(1,2);
maxVal=q(pos(1,1),pos(1,2));

for i=2:size(pos,1)
    aux=q(pos(i,1),pos(i,2));
    if aux>maxVal
        coor(1)=pos(i,1); coor(2)=pos(i,2);
        maxVal=q(pos(i,1),pos(i,2));
    end
end

% Pintamos el rectángulo rojo alrededor del patrón

        a(coor(1):coor(1)+np,coor(2)-2:coor(2)+2,1)=255;
        a(coor(1):coor(1)+np,coor(2)-2:coor(2)+2,2)=0;
        a(coor(1):coor(1)+np,coor(2)-2:coor(2)+2,3)=0;

        a(coor(1):coor(1)+np,coor(2)+mp-2:coor(2)+mp+2,1)=255;
        a(coor(1):coor(1)+np,coor(2)+mp-2:coor(2)+mp+2,2)=0;
        a(coor(1):coor(1)+np,coor(2)+mp-2:coor(2)+mp+2,3)=0;

        a(coor(1)-2:coor(1)+2,coor(2):coor(2)+mp,1)=255;
        a(coor(1)-2:coor(1)+2,coor(2):coor(2)+mp,2)=0;
        a(coor(1)-2:coor(1)+2,coor(2):coor(2)+mp,3)=0;

        a(coor(1)+np-2:coor(1)+np+2,coor(2):coor(2)+mp,1)=255;
        a(coor(1)+np-2:coor(1)+np+2,coor(2):coor(2)+mp,2)=0;
        a(coor(1)+np-2:coor(1)+np+2,coor(2):coor(2)+mp,3)=0;
    else
        maxVal=-1;
        oclu=1;
    end
end

im1=uint8(a);
```

Capítulo 8

Conclusiones

A lo largo de este proyecto hemos desarrollado un estudio sobre el reconocimiento de patrones en secuencias de vídeo mediante el uso de la desigualdad matemática de Cauchy-Schwartz. El estudio se ha dividido en dos fases claramente diferenciadas. En la primera parte y por completitud recordamos el análisis hecho en el año 2009, donde centrando primero nuestra atención en el caso unidimensional (aplicado a la búsqueda de patrones en señales de audio), y posteriormente en imágenes digitales que corresponde al caso dos dimensional, logramos buenos resultados en la búsqueda. En dicho estudio se aplicaron diferentes transformaciones sobre las señales de audio y las imágenes, y en casos prácticos se testó la robusted del algoritmo propuesto frente a dichas transformaciones. Como resultado dedujimos que el algoritmo de reconocimiento de patrones funciona de manera eficiente y robusta frente a las transformaciones consideradas.

En esta segunda fase, hemos realizado el estudio con secuencias de vídeo que corresponde al caso tres dimensional, donde una dimensión es la temporal, que presenta peculiaridades muy notables, como hemos podido observar en la experimentación. De hecho aunque teníamos esperanzas de obtener buenos resultados simplemente aplicando el filtro basado en la desigualdad de Cauchy Schwarz, nos hemos dado cuenta de que únicamente con este criterio el algoritmo no funcionaba correctamente en muchos casos. La transformación debida al movimiento de los fotogramas puede variar significativamente la escena y el patrón, y por tanto complicar mucho la búsqueda. Es por ello que nos hemos centrado en un caso simplificado, en el cual consideramos que no se producen oclusiones del patrón en la secuencia de vídeo. Incluso en este caso ha sido necesario añadir dos criterios adicionales, que

son la continuidad respecto del tiempo (entre fotogramas) de la posición del patrón dentro de la escena y la cercanía entre el valor medio de los píxeles del template y el de los píxeles de la muestra. Con este algoritmo hemos logrado unos resultados modestos pero que dejan entrever un posible camino para abordar el tema de una manera más amplia y más exhaustiva.

Por otra parte, hemos creado una agradable a la par que simple interfaz que permite ejecutar el algoritmo sin necesidad de entender los algoritmos Matlab y que muestra los resultados de seguimiento de un patrón en un vídeo dado.

Por último, el software desarrollado tanto en la versión unidimensional, como la bidimensional, y la de secuencias de vídeo ha sido publicada en abierto, lo que también supone una interesante aunque modesta contribución a la comunidad de software libre, que puede reutilizar este trabajo para cualquier otro fin relacionado. La dirección de consulta de dicha publicación es https: www.bib.upct.es.

Bibliografía

[1] J. Gomes, L. Velho. Computaçao Gráfica: Imagem. *Instituto de Matemática Pura e Aplicada (IMPA)*, 2002.

[2] E. Trucco y A. Verri. Introductory Techniques for 3-D Computer Vision.*Prentice Hall*,1998.

[3] J. Arvesú, R. Álvarez, F. Marcellán. Álgebra Lineal y Aplicaciones. *Síntesis*,Madrid, 1999.

[4] J. Burgos. Álgebra Lineal. *McGraw-Hill*,Madrid, 1993.

[5] A. Leger, T. Omachi, G. Wallace. The JPEG still Picture Compression Algorithm and its Applications. *Global Telecomunications Conference (IEEE)*, 1988.

[6] J.M. Miguel Jiménez. Laboratorio de Visión Artificial *Universidad de Alcalá de Henares*, 2007.

[7] A. Cordero, J.L. Hueso, E. Martínez, J.R. Torregrosa. Métodos Numéricos con Matlab. *Universidad Politécnica de Valencia*, 2005.

[8] A. Erhardt. Theory and Applications of Digital Image Processing. *Offenburg University of Applied Sciences*, 2004.

[9] G. Pajares, J. De la Cruz. Visión por Computador - Imágenes Digitales y Aplicaciones 2ª Ed. *Alfaomega*, 2008.

[10] J. García de Jalón, J.I. Rodríguez, J. Vidal. Aprenda Matlab como si estuviese en primero. *Universidad Politécnica de Madrid*, 2007 .

[11] S. Amat, F. Aràndiga , Cohen A. and R. Donat. Tensor product multiresolution analysis with error control for compact image representation. *Signal Processing*, **82**(4), 587-608, 2002.

[12] S. Amat, F. Aràndiga, A. Cohen, R. Donat, G. García and M. Von Oehsen. Data compression with ENO schemes: A case study. *Applied and Computational Harmonic Analysis*, **11**, 273-288, 2001.

[13] S. Amat, S. Busquier and J.C. Trillo. Stable Interpolatory Multiresolution in 3D. *Applied Numerical Analysis and Computational Mathematics*, **2**(2), 177-188, 2005.

[14] S. Amat, S. Busquier and J.C. Trillo. Non-linear Hartens's Multiresolution on the Quincunx Pyramid. *Journal of Computational and Applied Mathematics*, **189**, 555-567, 2006.

[15] S. Amat, R. Donat, J.Liandrat and J.C.Trillo. Proving convexity preserving properties of interpolatory subdivision schemes through reconstruction operators. In preparation.

[16] S. Amat, R. Donat, J. Liandrat and J.C. Trillo. Analysis of a new nonlinear subdivision scheme. Applications in image processing. *Foundations of Computational Mathematics*, **6**(2), 193-226, 2006.

[17] S. Amat and J. Liandrat. On the stability of the PPH nonlinear multiresolution. *Applied Computational Harmonic Analysis*, **18**(2), 198-206, 2005.

[18] F. Aràndiga and R. Donat. Nonlinear Multi-scale Decomposition: The Approach of A.Harten. *Numerical Algorithms*, **23**, 175-216, 2000.

[19] F. Aràndiga, R. Donat and A. Harten. Multiresolution Based on Weighted Averages of the Hat Function I: Linear Reconstruction Operators, *SIAM Journal on Numerical Analysis*, **36**, 160-203, 1999.

[20] F. Aràndiga, R. Donat and A. Harten. Multiresolution Based on Weighted Averages of the Hat Function II: Nonlinear Reconstruction Operators, *SIAM Journal on Scientific Computing*, **20**(3), 1053-1099, 1999.

[21] F. Aràndiga and J.C. Trillo. Resultados de Interpolación No Lineal. *Revista Matemàtica de la Universitat de València*, **1**(1).

[22] S. Bachelli and S. Papi. Filtered wavelet thresholding mehthods. *Journal of Computational and Applied Mathematics*, **164-165**, 39-52, 2004.

[23] A.S. Cavaretta, W. Dahmen and C.A Micchelli. Stationary subdivision. *Memoirs of the American Mathematical Society*, **93**(453):186, 1991.

[24] F. Chen. Estimating subdivision depths for rational curves and surfaces. *ACM Transactions Graphics*, **11**(2), 140-151, 1992.

[25] A. Cohen, I. Daubechies and J.C. Feauveau. Biorthogonal bases of compactly supported wavelets. *Communications in Pure and Applied Mathematics*, **45**, 485-560, 1992.

[26] A. Cohen, N. Dyn and B. Matei. Quasilinear subdivision schemes with applications to ENO interpolation. *Applied Computational Harmonic Analysis*, **15**, 89-116, 2003.

[27] G. Delauries and S. Dubuc. Symmetric Iterative Interpolation Scheme. *Constructive Approximations*, **5**, 49-68, 1989.

[28] D. Donoho. Interpolating wavelet transforms. *Prepint Standford University*, 1994.

[29] D. Donoho. Denoising by soft thresholding, *IEEE Transactions on Informations Theory*, **41**(3), 613-627, 1995.

[30] N. Dyn. Subdivision schemes in computer aided geometric design. *Advances in Numerical Analysis II., Subdivision algorithms and radial functions, W.A. Light (ed.), Oxford University Press*, 36-104. Prentice-Hall, 1992.

[31] N. Dyn, A. Gregory and D. Levin. A 4-point interpolatory subdivision scheme for curve design. *Computer Aided Geometric Design*, **4**, 257-268, 1987.

[32] N. Dyn, J.A. Gregory and D. Levin. Analysis of linear binary subdivision schemes for curve design. *Constructive Approximations*, **7**, 127-147, 1991.

[33] N. Dyn, F. Kuijt, D. Levin and R. Van Damme. Convexity preservation of the four-point interpolatory subdivision scheme. *Computer Aided Geometric Design*, **16**(8), 789-792, 1999.

[34] E. Fatemi, J. Jerome and S. Osher. Solution of the hydrodynamic device model using high order non-oscillatory shock capturing algorithms. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **10**, 232-244, 1991.

[35] M.S. Floater and C.A. Micchelli. Nonlinear stationary subdivision. *Approximation theory: in memory of A.K. Varna, edt: Govil N.K, Mohapatra N., Nashed Z., Sharma A., Szabados J.*, 209-224, 1998.

[36] J. Gomes and L. Velho. Computação Gráfica: Imagem, *Série Computação e Matemática*, 2002.

[37] A. Harten. ENO Schemes with subcell resolution, *Journal of Computational Physics*, **83**, 148-184, 1989.

[38] A. Harten. Discrete multiresolution analysis and generalized wavelets, *Journal Applied Numerical Mathematics*, **12**, 153-192, 1993.

[39] A. Harten. Multiresolution representation of data II, *SIAM Journal on Numerical Analysis*, **33**(3), 1205-1256, 1996.

[40] A. Harten, B. Engquist, S.J. Osher and S.R. Chakravarthy. Uniformly high order accurate essentially non-oscillatory schemes III, *Journal of Computational Physics*, **71**, 231-303, 1987.

[41] A. Harten, S.J. Osher, B. Engquist and S.R. Chakravarthy. Some results on uniformly high-order accurate essentially non-oscillatory schemes, *Applied Numerical Mathematics*, **2**, 347-377, 1987.

[42] G.-S. Jiang and C.-W. Shu, Efficient implementation of weighted ENO schemes. *Journal of Computational Physics*, **126**, 202-228, 1996.

[43] F. Kuijt and R. van Damme. Convexity preserving interpolatory subdivision schemes. *Constructive Approximations*, **14**, 609-630, 1998.

[44] A. Marquina. Local piecewise hyperbolic reconstruction of numerical fluxes for nonlinear scalar conservation laws. *SIAM Journal on Scientific Computing*, **15**(4), 892-915, 1994.

[45] C.A. Micchelli. Interpolatory subdivision schemes and wavelets. *Journal of Approximation Theory*, **86**, 41-71, 1996.

[46] G. Mustafa, F. Chen and J. Deng. Estimating error bounds for binary subdivision curves/surfaces. *Journal of Computational and Applied Mathematics*. To appear.

[47] X.-D. Liu, S. Osher and T. Chan, Weighted essentially non-oscillatory schemes. *Journal of Computational Physics*, **115**, 200-212, 1994.

[48] M. Rabbani and P.W. Jones. Digital Image Compression Techniques. Tutorial Text, *Society of Photo-Optical Instrumentation Engineers (SPIE)*, TT07, 1991.

[49] C.-W. Shu. Numerical experiments on the accuracy of ENO and modified ENO schemes. *Journal of Scientific Computing*, **5**, 127-149, 1990.