UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DEPARTAMENTO DE MATEMÁTICA APLICADA

DISEÑO DE FILTROS PARA EL PROCESADO DE IMÁGENES BASADOS EN TEORÍA DE GRAFOS

Cristina Pérez Benito

Tutores:

J.Alberto Conejero

Samuel Morillas

20 de julio de 2015

A **Alberto**, por su orientación, paciencia y aliento continuo y necesario.
A **Samuel**, por todo su conocimiento, tiempo y constante dedicación.
A **Cristina Jordán**, por su trabajo infatigable e inestimable.
A todos, GRACIAS.

Resumen

En este trabajo, se llevará a cabo el estudio de diferentes algoritmos de procesamiento de imágenes, tanto para segmentación como para eliminación de ruido Gaussiano basándonos en teoría de grafos.

La finalidad es generar una nueva herramienta automática que mejore la calidad de las imágenes. Para llevar a cabo esto partiremos de unas ciertas imágenes con un tipo concreto de ruido (ruido Gaussiano) y se desarrollaran diferentes algoritmos de filtrado de la imagen. Se buscará optimizar sus resultados aplicando diferentes técnicas que se explicarán a lo largo del trabajo.

El objetivo final es la mejora de unos ciertos métodos de filtrado ya existentes, de los que conocemos sus resultados y con los que podremos comparar basándonos para ello en las llamadas medidas de similitud que también serán explicadas en lo sucesivo.

Comenzaremos desarrollando los fundamentos del tratamiento de imágenes, así como los tipos de ruido existentes y los posibles métodos de procesamiento tanto para reducción de ruido como para segmentación. En el siguiente capítulo se introducirán los conceptos de teoría de grafos necesarios para el desarrollo del trabajo.

En el último capítulo, se presentará la nueva herramienta de segmentación de imágenes, que permitirá distinguir zonas de detalle y zonas homogéneas en una imágen con gran eficiencia y como con ella se pueden mejorar algunos filtros ya conocidos en este área. Esta idea también será utilizada para la implementación de un nuevo método de filtrado de imágenes basado en teoría de grafos.

Tras conocer el funcionamiento de estas nuevas herramientas, procederemos a ver los resultados que proporcionan y compararlos con otros métodos ya existentes para conocer su rendimiento. Esta comparación se realizará, como decíamos, mediante las llamadas medidas de similitud, nos apoyaremos en el PSNR (Peak Signal-to-Noise Ratio), por ser la medida más utilizada históricamente, y en una nueva medida denominada FCSS (Fuzzy Colour Structural Similarity), por ser más fiel a la percepción humana.

Índice general

Capítulo 1

Introducción

El Procesamiento Digital de Imágenes es el conjunto de técnicas que se aplican a las imágenes digitales con la intención de mejorar o modificarlas. El objetivo final de este tipo de técnicas es facilitar el almacenamiento, la duplicación o la extracción de información.

Las imágenes digitales son: fotografías, documentos, manuscritos, textos e ilustraciones escaneadas u obtenidas en forma digital, donde se ha confeccionado un mapa de ellas en forma de cuadrícula, cuyos puntos son los llamados píxeles. A cada píxel se le asigna un valor de tono (negro, blanco, matices de grises o de color) y se almacena secuencialmente en la memoria, donde se reduce a una representación matemática.

En la bibliografía especializada en el tema se puede encontrar una gran variedad de formas de clasificación de las técnicas utilizadas en el tratamiento de imágenes, un resumen de dicha clasificación se muestra a continuación:

- Mejora de la calidad.
- Técnicas de restauración.
- Compresión.
- Segmentación.

Orígenes

Los orígenes del procesado de imágenes digitales se remontan a la impresión de periódicos en 1921. En aquella época la codificación y transmisión de datos se realizaba mediante un cable submarino entre Londres y Nueva York, donde se reconstruían e imprimían. Al año siguiente el proceso comenzó a mejorar gracias a una técnica basada en la reproducción fotográfica a través de cintas perforadas en las terminales telegráficas que permitían obtener 5 niveles de gris. Está técnica se fue mejorando hasta llegar a 5 niveles de gris en 1929.

Sin embargo, estas técnicas no se consideran los inicios del procesamiento digital de imágenes, puesto que no se utilizaba la computadora. Hubo que esperar hasta los años 60, con la llegada de los grandes ordenadores y el programa espacial estadounidense, para ver las primeras técnicas de procesado digital de imágenes por ordenador. Concretamente hasta el año 1964 en el Laboratorio de Propulsión de la NASA, cuando se procesaron las imágenes de la Luna enviada por el satélite Ranger 7, primera misión con éxito de Estados Unidos para explorar la Luna.

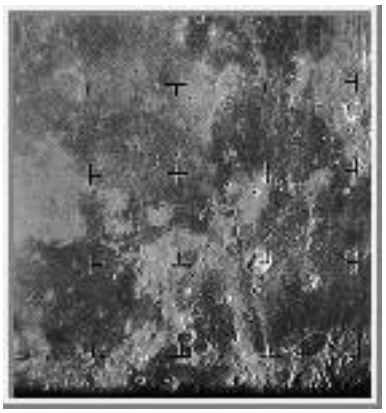


Figura 1.1: Primera imagen de la superficie lunar, NASA.

En las década de los 50 y 60 del pasado siglo, los Estados Unidos, así como la antigua Unión Soviética, iniciaron programas espaciales con la misión principal de la obtención de imágenes de la Tierra y el Sistema Solar para facilitar futuros viajes espaciales (Ranger, Luna Orbiter, Mariner). Se comprobó que muchas de aquellas imágenes sufrían importantes degradaciones debido a las difíciles condiciones de la adquisición de las mismas: vibraciones de la maquinaria, movimientos

de rotación, etc. Todo ello significaba perder información muy valiosa además de la enorme inversión que suponían estos programas. Esta situación motivó la aparición de un nuevo campo científico denominado procesamiento digital de imágenes.

La puesta en órbita del telescopio Hubble en 1990 dio origen a una de las aplicaciones más destacadas de este campo. El espejo primario del telescopio fue pulido con un dispositivo defectuoso, lo que provocó una generación de imágenes desenfocadas. Este suceso supuso una pérdida de casi 2.000 millones de dólares. Sin embargo, la comunidad científica reaccionó con una enorme cantidad de material sobre algoritmos de restauración, aportando soluciones a los defectos de las imágenes del telescopio.

A partir de este momento, el tratamiento de imágenes no ha cesado de crecer y beneficiarse de los continuos avances tecnológicos, entre los que se encuentra el desarrollo de lenguajes de programación de alto nivel, la invención del transistor o del circuito Integrado, desarrollo de los sistemas operativos, o la introducción del ordenador personal en 1981.

1.1. Motivación

Todo ser humano se encuentra rodeado de imágenes, la mayoría de la información la recibimos en forma de imágenes de todos tipo, en todos los colores y en blanco y negro. Es por esto que todo lo concerniente a la imagen tratada mediante un ordenador ha cobrado una enorme importancia durante los últimos tiempos.

Actualmente, son numerosas las áreas de aplicación involucradas en el procesamiento de imágenes digitales, con el objetivo de mejorar la calidad de la imagen para una correcta interpretación humana. Debido a ello, este área de la ciencia y la tecnología ha crecido enormemente durante los último años.

El procesamiento digital de imágenes consiste en el almacenamiento, procesamiento y representación de información de imágenes digitales por medio de una computadora.
El término imagen se refiere a una función bidimensional de intensidad de luz $f(x, y)$ donde x e y denotan las coordenadas espaciales. Una imagen digital pue-

de considerarse como una matriz cuyos índices identifican un punto de la imagen y el correspondiente valor del elememento de la matriz que identifica el nivel de intensidad de luz en ese punto.

Las imágenes digitales están sujetas a una amplia variedad de distorsiones durante la adquisición, procesamiento, compresión, almacenamiento, transmisión y reproducción, cualquiera de ellos puede dar lugar a una degradación de la calidad visual.

Para aplicaciones en las que en, última instancia, las imágenes son para ser vistas por seres humanos, el único método "correcto"de cuantificación de la calidad de una imagen visual es la evaluación subjetiva, sin embargo, en la práctica, la evaluación subjetiva es en general demasiado incómoda, lenta y cara. El objetivo de este área de investigación es el desarrollo de medidas cuantitativas para medir la calidad de la imagen de manera automática.

Áreas de aplicación

Hoy en día existe una inmensa gama de áreas involucradas en el procesamiento de imágenes digitales, con el objetivo de mejorar la calidad de la imagen para una correcta interpretación humana. Un criterio de clasificación habitual para diferenciar cada una de las áreas implicadas es el de la fuente de la imagen. La principal fuente de energía de las imágenes es el espectro electromagnético, pero existen otras como son la acústica, la ultrasónica, etc.

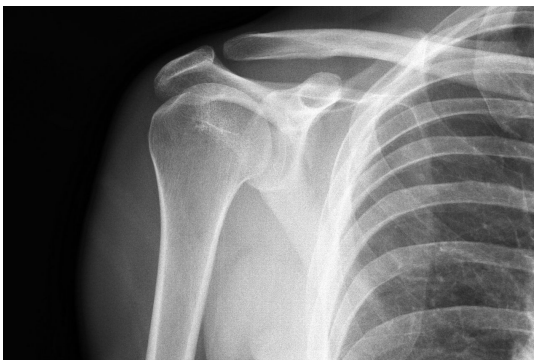


Figura 1.2: Imagen Rayos X.

Existen aplicaciones en medicina nuclear, como la tomografía por emisión de positrones, donde se le inyecta al paciente un isótopo radioactivo que emite rayos gamma y posteriormente son capturados. También es posible obtener imágenes empleando la banda ultravioleta del espectro electromagnético, teniendo como principales aplicaciones la litografía, la microscopía o los láseres.

Las imágenes captadas en banda visible e infrarrojas son, por mucho, las aplicaciones más numerosas. Se utilizan habitualmente en microscopía, astronomía, controles de calidad en la industria, etc.

Figura 1.3: Imagen infrarrojo.

Otras formas de obtener imágenes es a través del sonido, y dentro de este tipo lo más habitual es el ultrasonido, que tiene infinidad de aplicaciones médicas como puede ser el seguimiento de un embarazo, imágenes IVUS (intravascular ultrasound) utilizadas para diagnóstico de enfermedades cardiovasculares, etc.

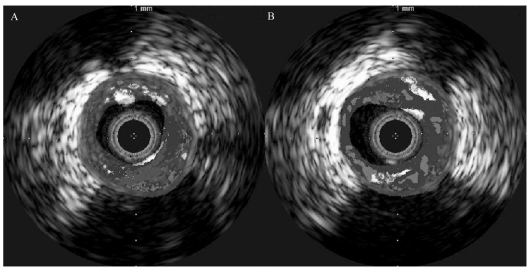


Figura 1.4: IVUS

La variedad, tanto de tipos de imagen como de métodos de tratamiento de las mismas, ha permitido explotar este campo de la ciencia en innumerables áreas:

- Análisis de materiales
  - Detección de impurezas.
  - Detección de fallas estructurales: grietas, cambios de densidad,...
- Tratamiento de documentos
  - Reconocimiento automático de caracteres.
  - Validación de firmas.
- Militar
  - Búsqueda automática de objetivos militares a partir de imágenes satélite.
  - Sistema de guía de misiles.
- Biología
  - Análisis de tejidos y células.
  - Clasificación de ADN.
- Biometría
  - Extracción de minucias y características para identificación biométrica.

1.2. Fundamentos tratamiento de imágenes

En este apartado presentaremos los conceptos básicos que se utilizan en el tratamiento de imágenes. Desarrollaremos los métodos básicos de filtrado, introduciremos el concepto de segmentación, ruido y algunas de las técnicas que se utilizan en este contexto. Por último hablaremos de las medidas de similitud que nos permiten comparar imágenes.

Una imagen analógica puede ser representada de forma aproximada por una serie de muestras igualmente espaciadas, este proceso se conoce como discretización; una imagen digital es la discretización tanto en coordenadas como en todos de gris de una imagen analógica. De esta manera, una imagen puede ser definida por una función $f(x, y)$, donde los valores x, y son coordenadas espaciales y el valor de la función f en dicho punto es conocido como intensidad o nivel de gris; se habla de imagen digital cuando x, y y los valores de f son cantidades finitas y discretas.

Definición 1.2.1. *Una imagen en tono de grises se define como una función*

$$f(x, y) : \Omega \subset \mathbb{R}^2 \longrightarrow \{0, \ldots, 255\}$$

donde $f(x, y)$ corresponde a la intensidad positiva de cada punto (x, y) de la imagen.

Una imagen digital es una representación bidimensional de una imagen a partir de una matriz numérica compuesta por un número finito de elementos, cada uno de los cuales tiene una localización y un valor. Estos elementos se conocen como píxel (acrónimo del inglés *picture element*, elemento de imagen). Es común encontrar que los valores sean entre 0 y 255, de manera que 0 indicaría la falta total de ese canal y 255 la intensidad máxima.

La representación más usual es como una matriz tridimensional $n \times m \times z$, donde n y m representan las dimensiones de la imagen y z es el número de canales o colores de la imagen.

Cuando existe un único canal se habla de imágenes en escala de grises, y cuando existe más de uno, de imágenes en color. Las operaciones sobre imágenes en escala de grises se pueden aplicar también a imágenes en color procesando cada canal por separado.

Existen varios tipos de imágenes. A continuación vemos algunas de ellas:

- **Imágenes de intensidad o en escala de grises**: son matrices de datos cuyos valores representan una escala determinada de intensidad, generalmente tendrán valores numéricos entre 0 y 255.
- **Imágenes binarias**: imagen digital con todos los valores de sus píxeles comprendidos entre 0 y 1.
- **Imágenes RGB (Red, Green and Blue)**: arrays de píxeles de color de tamaño $m \times n \times 3$, donde cada píxel esta formado por una terna de valores correspondientes a las componentes roja, verde y azul de la imagen en una posición determinada. Cada uno de los canales tendrá valores generalmente entre 0 y 255.

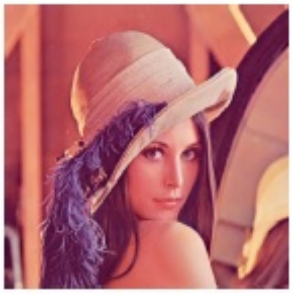


Imagen RGB sin descomponer

Canal Rojo

Canal Verde

Canal Azul

Existen otros modos de color a parte del RGB, como son el HSB (tono, saturación y brillo) o CMYK (cián, magenta, amarillo y negro), sin embargo, en este trabajo nos centraremos en RGB.

La **resolución** es la magnitud que relaciona los dos tamaños de la imagen, el físico y el digital. Suele medirse en términos de píxeles por pulgada (ppi) y de ella depende tanto la calidad de la representación como el tamaño que ocupa en memoria el archivo gráfico generado.
Dependiendo del número de píxeles la imagen digital poseerá más o menos resolución espacial.

Figura 1.5: Comparación imagen con distinto número de píxeles

Uno de los objetivos cuando se trabaja con filtrado de imágenes independientemente del ruido presente, es tener un buen equilibrio entre el suavizado de la imagen y la preservación de los bordes. El filtro ideal para la mejora de imágenes es aquel que es capaz de conservar bordes y detalles finos. Algunos filtros, al reducir el ruido, inevitablemente eliminan una cantidad considerable de detalles.

Por esta razón, entre otras, un problema fundamental del análisis de imágenes es el particionamiento de una imagen en regiones que son, en algún sentido, homogéneas y diferentes de las regiones vecinas. Esta homegeneidad puede ser medida con diferentes propiedades tales como el matiz, la sombra, la textura o el contraste.
Comenzaremos introduciendo el concepto de ruido y los tipos de ruido más usuales.

1.3. Ruido

El ruido en una imagen es un fenómeno muy frecuente que aparece de improvisto en la adquisición y/o transmisión por varias circunstancias, afectando a la calidad de la imagen. En el caso específico de las imágenes digitales lo definimos como los píxeles “erróneos", aleatorios que se entremezclan con los píxeles “aceptables"que componen la imagen y que entorpecen su correcta reproducción.

Reducir el ruido de una imagen es un tema que ha sido extensamente estudiado en el campo de procesamiento digital de imágenes.

Todos los procesos de captura de imagen están sujetos a ruido de algún tipo, hablaremos, por tanto, de los diferentes tipos de ruido en esta sección, para posteriormente hablar de las técnicas para eliminarlos.
Aunque existen muchos tipos de ruido, véase [9] y [8], veremos los más comunes y con especial interés en el ruido Gaussiano que será con el que tratemos.

Gaussiano

Entre los diferentes tipos de ruido, el más común es el ruido Gaussiano, que aparece en el momento de la adquisición de la imagen por un dispositivo en malas condiciones, mala iluminación o altas temperaturas y tiene como consecuencia el emborronamiento de todos los píxeles de la imagen.

En el ruido de tipo Gaussiano, la intensidad de todos y cada uno de los píxeles que componen la imagen se ven afectados y cambian su valor, de acuerdo con una distribución normal. Podría aplicarse otro tipo de distribución, sin embargo, se toma como modelo la gaussiana debido al teorema central del límite que nos asegura que la suma de diferentes ruidos tiende a aproximarse a una distribución normal.

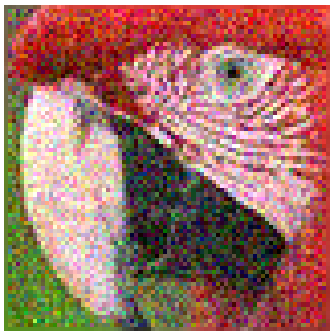
Se dice que un vector aleatorio $X = (X_1, X_2)$ sigue una distribución normal de parámetros μ_1,μ_2, σ_1, σ_2, σ_{12} y se denota por $X \sim N(\sigma, \mu)$, si su función de densidad conjunta viene dada por

$$f(X_1, X_2) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \exp\left\{-\frac{1}{2(1-\rho^2)}\left[\left(\frac{X_1-\mu_1}{\sigma_1}\right)^2 + \left(\frac{X_2-\mu_2}{\sigma_2}\right)^2 - 2\rho\frac{(X_1-\mu_1)(X_2-\mu_2)}{\sigma_1\sigma_2}\right]\right\}$$

siendo $\rho = \dfrac{cov(X_1, X_2)}{\sigma_1\sigma_2}$.

El valor final del píxel es el valor ideal más una cierta cantidad de error.

(a) Imagen original



(b) Imagen ruidosa, $\sigma = 30$

Ruido impulsivo

Es el más usual y se presenta durante la transmisión de datos por un canal contaminado. Los errores afectan a ciertos píxeles de la imagen (a diferencia del ruido gaussiano que afecta a todos), dejándolos completamente blancos o completamente negros. Se encuentra en situaciones de transito rápido, tales como defectos en los interruptores, causas naturales como rayos en la atmósfera, etc.

Este ruido también es frecuentemente llamado como “sal y pimienta". Se caracteriza por una densidad d, que representa una proporción entre el número de píxeles afectados y el tamaño de la imagen.

Figura 1.6: Imagen original e imagen con ruido impulsivo

Ruido uniforme

Es quizá el tipo de ruido menos descriptivo en situaciones prácticas, sin embargo, la densidad uniforme es bastante útil como base para numerosas generaciones de números aleatorios que se utilizan en las simulaciones.

La función de densidad de este tipo de ruido viene dado por

$$f(x,y) = \begin{cases} \dfrac{1}{(b-a)(d-c)} & a \leqslant x \leqslant b; c \leqslant y \leqslant d \\ \\ 0 & \text{en caso contrario} \end{cases}$$

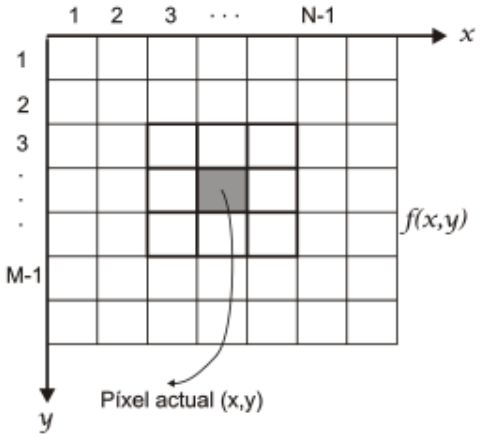
Capítulo 2

Estado del arte

Como ya hemos dicho anteriormente, una imagen puede ser representada a través de una matriz $m \times n$ de intensidades

$$\begin{pmatrix} f(1,1) & f(1,2) & \dots & f(1,n) \\ f(2,1) & f(2,2) & \dots & f(2,n) \\ \dots & \dots & \dots & \dots \\ f(m,1) & f(m,2) & \dots & f(m,n) \end{pmatrix}$$

donde $f(x,y)$ es la intensidad del píxel situado en la posición (x,y).



Dentro del procesamiento digital de imágenes, existen diversas ramas: eliminación de ruido, compresión de imágenes, segmentación, detección de bordes

etc. En la actualidad existe infinidad de métodos para abordar todos cada una de estas áreas, [1], [2], [21].
En este capítulo veremos brevemente algunas de las técnicas más conocidas para eliminación de ruido, segmentación y medidas de similitud entre imágenes.

Se denotará en lo sucesivo, a cada píxel de la imagen I con su posición (x, y), y a la intensidad del píxel i situado en la posición (x, y) de la imagen I como $I(x, y)$, aunque en muchas ocasiones, por simplicidad, se denotará a cada píxel sólo por x_i.

2.1. Filtrado de Imágenes

El ruido presente en una imagen se puede reducir por medio de filtros basados en el domino espacial o de frecuencia [1]. Los filtros de dominio en frecuencia se basan en la utilización de la transformada de Fourier de una imagen, modificando esta para lograr un objetivo específico y después calcular la transformada inversa para obtener la imagen procesada. La transformada de Fourier se ve como una transformación de un dominio (espacial) a otro dominio (frecuencias) sin perder la información de la imagen. Dentro de los filtros de este tipo se pueden mencionar los filtros de paso bajo y de paso alto que se utilizan para atenuar las frecuencias bajas y las altas respectivamente.

Por otro lado, los filtrados de dominio espacial se llevan a cabo directamente sobre los píxeles de la imagen. Estos filtros consideran una vecindad al píxel que se está procesando y una operación predefinida que se realiza sobre los píxeles abarcados por la vecindad. El filtro creará un nuevo píxel que se intercambiará por el píxel central de la vecindad escogida. El grado de filtrado sobre la imagen dependerá de la cantidad de píxel vecinos que se estén utilizando.

En este trabajo se desarrollarán filtrados espaciales, que ofrecen buenos resultados en calidad para eliminar el ruido gaussiano, que será en el que nos centremos. Dentro de los filtros espaciales, aunque existen muchos más, los más comunes son los filtros de suavizado, que se utilizan para hacer que la imagen aparezca algo borrosa y también para reducir el ruido. Hacer que la imagen aparezca algo borrosa puede ser útil en las etapas de preprocesado, por ejemplo en la eliminación de los detalles antes de la extracción de un objeto grande y en el relleno de

pequeños espacios entre líneas o curvas antes de la extracción de contornos. Dentro de este tipo de filtros podemos encontrar:

- Filtros lineales, o filtros de promedio: filtro de media aritmética, media geométrica,...
- Filtros no lineales: filtro de mediana, filtro de punto medio y media,...

Estos filtros son muy utilizados para la reducción de ruido en escala de grises. Las primeras soluciones para filtrar una imagen de color se trataba de aplicar las mismas técnica a cada canal de color. Sin embargo, pueden aparecer artefactos de color y otros efectos que requieren otros filtros más apropiados.
Una de las vertientes más estudiadas para el tratamiento de imágenes en color es el filtrado vectorial, considerando cada píxel como un vector formado por las componentes de color.

En ocasiones, filtrar la imagen entera supone procesar píxeles no ruidosos y con ello un suavizado excesivo. Para arreglar este problema, aparecen los filtros que se adaptan a las características locales de la imagen, por ello denominados *filtros adaptativos*. Entre ellos, cabe destacar por la buena calidad de imagen filtrada que proporcionan, los filtros en los que solo se procesan los píxeles que se detectan ruidosos, dejando los demás píxeles iguales. El concepto de *peer group* es uno de los métodos actuales más estudiados para la detección de píxeles erróneos, [17].

2.1.1. Filtrado espacial

Todas las imágenes contienen detalles, compuesto por transiciones de intensidad que varían en ciclos que van del oscuro al claro. La tasa a la cual la intensidad varía completando un ciclo es su frecuencia espacial.

Una imagen está formada por componentes de frecuencia que varían de bajas frecuencias a altas frecuencias, donde hay transiciones rápidas de intensidad, hay frecuencias espaciales altas, mientras que en transiciones que cambian lentamente representan frecuencias bajas. Las altas frecuencias en una imagen aparecen representando bordes o detalles. Una imagen puede filtrarse para acentuar o eliminar una banda de frecuencias, tales como las altas frecuencias o las bajas. Estas

operaciones de procesamiento digital de imágenes se conocen como *filtrado espacial*.

El procesamiento espacial opera sobre un grupo de píxeles que rodean a un píxel central, sustituyendo a este por algún tipo de combinación de sus píxeles adyacentes. La reducción de ruido se puede lograr mediante filtros con enfoque lineal y también con filtros no lineales.

Filtrados espaciales lineales

El filtrado lineal o también llamado de pasos bajos, puesto que mantiene intocables las frecuencias bajas, [2], tiene por objeto suavizar los contrastes de la imagen, de modo que las componentes de alta frecuencias quedan atenuados.

La salida de este tipo de filtros no es más que un promedio de los píxeles contenidos en la vecindad de un punto determinado de la imagen que se quiere procesar, se elige normalmente vecindades de 3×3 ó 4×4 que llamaremos ventanas.

La construcción más simple consiste en una mácara en la que todos los coeficientes son 1 y el resultado se divide entre el número de elementos de la máscara (es un promedio móvil).

$$\frac{1}{9}\times\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix} \qquad \frac{1}{16}\times\begin{bmatrix}1&2&1\\2&4&2\\1&2&1\end{bmatrix}$$

Figura 2.1: Máscara 3×3 filtros pasos bajos

El filtro de paso bajo más sencillo es el **filtro de media aritmética**.

Sea W una ventana de tamaño $n \times n$ centrado en x_i. El filtro de media aritmética calcula el valor promedio de la ventana W y sustituye el píxel central

por dicho valor. Es decir, el nuevo valor de x_i viene dado por

$$FMA_{x_i} = \frac{1}{n^2} \sum_{x_j \in V} x_j$$

Este filtro se utiliza directamente sobre la imagen ruidosa o después de un proceso de detección de píxeles ruidosos. En caso de estar ante una imagen RGB, se aplica el mismo método para cada uno de los 3 canales.

Filtrados espaciales no lineales

Existen filtros que no son función lineal de los valores de intensidad de los píxeles de una imagen. Es decir, no se calculan como una suma lineal de elementos multiplicados por pesos constantes. Estos filtros se conocen como *filtros no lineales*. Constituyen también técnicas de procesamientos por grupo de píxeles, operando sobre un núcleo de píxeles de entrada que rodean al píxel central. La diferencias es que en lugar de utilizar un promedio ponderado, se utilizan otras técnicas que combinan los valores del grupo de píxeles de entrada.

Uno de los filtros de suavizado no lineales más conocido es el **filtro de mediana**.

Una de las características del filtro de media es que difumina los bordes y otros detalles de la imagen. Cuando el objetivo es más la reducción de ruido que el difuminado, el empleo de filtros de mediana representa una posibilidad alternativa.

Este filtro crea nuevos píxeles en una nueva imagen, esto se realiza calculando la mediana del conjunto de píxeles del entorno, en lugar de la media. De esta forma se homogeneízan los píxeles de intensidades muy diferentes con respecto a los vecinos. Este método es particularmente efectivo cuando el ruido consiste de componentes fuertes y de forma picuda, y la característica que se desea preservar es la agudeza de los bordes.

Este filtro se basa en reemplazar el valor del píxel central por la mediana de los niveles de intensidad de los píxeles de su ventana, es decir

$$x_i = \text{mediana}_{x_j \in V}\{x_j\}$$

Existen otros filtros no lineales, como el filtro de la moda, que pueden consultarse en [1], [2].

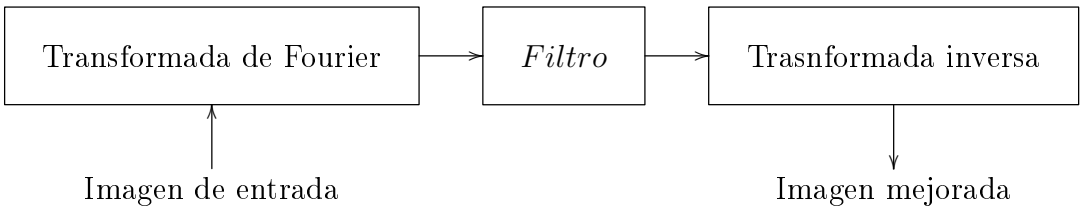
2.1.2. Filtrado en el dominio de frecuencia

Este tipo de filtrado se basan en la transformada de Fourier, que permite expresar una función como una integral de senos y cosenos, y en la posibilidad de realizar la transformada inversa sin pérdida de información.

Debido a que las imágenes a tratar son de tipo discreto, interesa trabajar con la transforma Discreta de Fourier.

Definición 2.1.1. *Sea I una imagen de tamaño $m \times n$, se define la transforma discreta de Fourier en dos dimensiones como*

$$\hat{f}(u,v) = \frac{1}{mn} \sum_{x=0}^{m-1} \sum_{y=0}^{n-1} f(x,y) e^{-j2\pi(ux/m+vy/m)}$$



La función de transferencia de filtrado, que denotamos por $H(x,y)$ y se define como

$$H(x,y) = \frac{I(x,y)}{\widehat{I}(x,y)}$$

donde I es la imagen a filtrar e $\widehat{I}$ es la salida, es la que actúa sobre la transformada de la imagen y permite suprimir ciertas frecuencias mientras que deja otras invariantes.

Las frecuencias altas son las responsables de los bordes o el ruido de la imagen, mientras que las frecuencias bajas representan zonas homogéneas. Esto

permite hablar de dos tipos de filtros, los llamados *filtros pasobajos* que permiten atenuar las altas frecuencias y dejar invariantes las bajas, y los *filtros pasoaltos* que atenúan bajas frecuencias y mantienen las altas.

Este tipo de filtros, vienen caracterizados por una frecuencia de corte, D_0, que permitirá hacer un mayor o menor suavizado. Los función de trasferencia de los filtros se encargará de cortar las frecuencias que superen o que queden por debajo de esta frecuencia de corte. A continuación veremos, sin entrar en mucho detalle, los filtros más relevantes.

Ideal

El filtro pasabajos ideal se define por:

$$H(x,y) = \begin{cases} 1 & si \quad D(x,y) \leq D_0 \\ \\ 0 & si \quad D(x,y) > D_0 \end{cases}$$

donde

$$D(x,y) = [(x - m/2)^2 + (y - n/2)^2]^{1/2}$$

De manera análoga podemos definir el filtro Pasaaltos ideal:

$$H(x,y) = \begin{cases} 1 & si \quad D(x,y) > D_0 \\ \\ 0 & si \quad D(x,y) \leq D_0 \end{cases}$$

Filtro Butterworth

La función de transferencia del filtro de Butterworth Pasabajos (BLPF, *Butterworth Low Pass Filter*), de orden s con frecuencia de corte D_0 es:

$$H(x,y) = \frac{1}{1 + [D(x,y)/D_0]^{2s}}$$

y de manera análoga podemos definir el filtro Pasaaltos (BHPF, *Butterworth High Pass Filter*):

$$H(x,y) = \frac{1}{1 + [D_0/D(x,y)]^{2s}}$$

Filtro Gaussiano

La función de transferencia del Filtro Gaussiano Pasobajos (GLPF, *Gaussian Low pass Filter*) con frecuencia de corte D_0 esta dado por:

$$H(x,y) = e^{-D^2(x,y)/2D_0^2}$$

Por otro lado, La función de transferencia del Filtro Gaussiano Pasaaltas (GHPF, *Gaussian High pass Filter*) con frecuencia de corte localizada a una distancia D_0 desde el origen esta dado por:

$$H(x,y) = 1 - e^{-D^2(x,y)/2D_0^2}$$

2.2. Proceso de detección

La mayoría de los métodos para identificar los píxeles erróneos de una imagen consisten en calcular la distancia entre un píxel y sus vecinos, y etiquetarlo como ruidoso cuando dicha distancia supere un cierto umbral para la mayoría de los píxeles vecinos con que se ha comparado.

Existen diferente métricas, algunas en términos de la intensidad de dos píxeles, otras en función del ángulo que pueden formar dos píxeles, etc. A continuación se explican algunas de estas métricas.

Distancia euclídea

La distancia euclídea entre dos píxeles x_i, x_j se define como:

$$d_2(x_i, x_j) = \| x_i - x_j \|_2$$

Consideraremos esta distancia para trabajar en imágenes RGB, en caso de imágenes en escala de grises se reduce a

$$d_2(x_i, x_j) = | x_i - x_j |$$

Métrica del coseno del ángulo

La distancia del ángulo del coseno entre dos píxeles x_i, x_j,

$$C(x_i, x_j) = \frac{x_i^t x_j}{\| x_i \|_2 \| x_j \|_2}$$

donde $x_i \neq 0$, $x_j \neq 0$, se considera el rango de los píxeles de $[1, 256]$.

Métricas fuzzy

Una métrica fuzzy estacionaria M sobre una imagen I es un conjunto fuzzy de $I \times I$ satisfacciendo $\forall\, x,y,z \in I$:

1. $M(x, y) > 0$
2. $M(x, y) = 1 \Leftrightarrow x = y$
3. $M(x, y) = M(y, x)$
4. $M(x, z) \geqslant M(x, y) * M(y, z)$ donde $*$ es una continuidad de t-norma.

$M(x, y)$ representa el grado de cercanía de los píxeles x e y, $M(x, y)$ es cercano a 0 cuando x está lejos de y.

Recordemos que las t-normas fueron introducidas por Schweizer y Sklar en [22] en el marco de los espacios métricos probabilísticos y se definen como una operación binaria:

$$* : [0, 1]^2 \longrightarrow [0, 1]$$

que cumple las siguientes propiedades:

- Conmutativa: $x * y = y * x$
- Asociativa: $(x * y) * z = x * (y * z)$
- Monotonicidad: si $x \leq y$ y $w \leq z \Rightarrow x * w \leq y * z$
- Condiciones de frontera: $x * 0 = 0$, $x * 1 = x$

Peer group

En una imagen I, el peer group $P(x_i, d)$ asociado a un píxel x_i, consiste en los x_j píxeles en una ventana W de tamaño $n \times n$ centrada en x_i que están más cercanos en intensidad a x_i medida con una cierta distancia d. A partir de los x_j píxeles del peer group, se puede determinar si x_i es erróneo o no.
Se conoce como PGA (Peer group averaging) al proceso de reemplazar $I(x_i)$ por la media de su peer group. Para calcular la distancia entre los píxeles, se pueden utilizar cualquiera de las medidas expuestas anteriormente. A la hora de asociar un peer group a un píxel, se ha de tener en cuenta dos parámetros:

- Tamaño de la ventana: n.

- Número peer group: N.

La elección de estos parámetros se puede realizar mediante distintos métodos, como por ejemplo, el discriminante de Fisher, véase [17].

2.3. Segmentación

El primer paso en cualquier proceso de análisis de imagen es la segmentación. La segmentación consiste en dividir la imagen en las partes que lo forman, basándose en ciertas características locales que nos permiten distinguir un objeto del fondo y objetos entre sí. El nivel al que se realiza esta subdivisión dependerá de la aplicación particular que se vaya a llevar a cabo, la segmentación terminará cuando se hayan detectado todos los objetos de interés para la aplicación. En general, la segmentación automática es una de las tareas más complicadas dentro del procesado de imágenes.

La mayoría de las imágenes están constituidas por regiones o zonas que tienen características homogéneas. Generalmente estas regiones corresponden a objetos de la imagen. La segmentación consistirá en la partición de la imagen en varias zonas o regiones homogéneas y disjuntas a partir de su contorno, conectividad, o en términos de un conjunto de caracteríscticas de los píxeles de la imagen que permitan discriminar unas regiones de otras.

Los tonos grises, los momentos, la magnitud del gradiente, la dirección de los bordes, etc., son caracteríscticas a utilizar para la segmentación.

Hoy en día existen multitud de técnicas de segmentación de una imágen

- **Segmentación basada en el valor del píxel**: dentro de este grupo se consideran técnicas de agrupamiento de píxeles en un espacio de color, de acuerdo a alguna característica, los algoritmos de agrupamiento difuso y la umbralización.
- **Segmentación mediante la detección de contornos**: se realiza mediante la búsqueda de discontinuidades en la imágenes, es decir, de píxeles en los cuales el nivel de gris varíe bruscamente respecto al de sus vecinos,

[10]. Podemos destacar en este tipo de métodos la búsqueda de puntos de elevado gradiente, el algoritmo de Canny ([11]), etc.

- **Segmentación basada en modelos físicos:** permite segmentar mediante el estudio del proceso de reflexión de luz y formación de la imagen. Una de las técnicas más usuales es el crecimiento de Regiones (Region Growing, [12]) debido a su alta eficiencia computacional.

A continuación se muestran dos métodos sencillos de segmentación, uno basada en el valor de píxel y otra en la detección de contornos:

Detección de puntos

Un punto asilado de una imagen tiene un tono gris que difiere significativamente de los tonos de gris de sus píxeles adyacentes.

Veamos como funciona este tipo de métodos con un ejemplo:

Ejemplo 2.3.1. $M =$

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

definimos

$$g(x,y) = -f(x-1,i+1) - f(x-1,y) - f(x+1,y+1) - f(x-1,y) +$$

$$8f(x,y) - f(x+1,y) - f(x-1,y-1) - f(x,y-1) - f(x+1,y-1)$$

-1	-1	-1
-1	8	-1
-1	8	-1

(a) Máscara

1	1	1	1	1
1	10	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

(b) Imagen Original

-	-	-	-	-
-	72	-9	0	-
-	-9	-9	0	-
-	0	0	0	-
-	-	-	-	-

(c) Imagen filtrada

De este modo diremos que el píxel situado en (x, y) es un punto aislado si

$$\mid g(x,y) \mid > T$$

donde T es un valor umbral fijado anteriormente, que dependerá de la aplicación que estemos realizando.
Sin embargo, este filtro podría detectar como puntos aislados píxeles que formen parte de un borde, por ello es más conveniente utilizar el siguiente filtro no lineal:

$$R(x,y) = \min_{(x,y)\neq(r,s)} \mid f(r,s) - f(x,y) \mid$$

y diremos que el píxel situado en (x, y) es un punto aislado si

$$\mid R(x,y) \mid > T$$

Detección de contorno

La detección de contornos es una de las técnicas mas comunes para detectar discontinuidades significativas en los valores de intensidad de una imagen. Estas discontinuidades son detectadas usando la primera y segunda derivada. El gradiente de una función f se define como:

$$\bigtriangledown f = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \\ \frac{\partial f}{\partial y} \end{pmatrix}$$

Se utiliza una aproximación del gradiente para realizar los cálculos con el fin de evitar el número de operaciones:

$$\bigtriangledown \approx \mid \frac{\partial f}{\partial x} \mid + \mid \frac{\partial f}{\partial y} \mid$$

La propiedad fundamental del vector gradiente es que siempre apunta en la dirección del índice máximo de variación de la función. Un contorno se caracteriza por tener una transición de claro a oscuro o viceversa. Esta transición se puede detectar calculando el gradiente de la imagen en dos direcciones ortogonales. Los operadores gradientes más comunes son los operadores de Roberts y Sobel, [3], [4].

Una de las plantillas detecta contornos horizontales y la otra los verticales, obteniéndose así dos imágenes con dos gradientes: I_{g_1}, I_{g_2}. Para definir si en un píxel determinado hay un contorno o no, se define un umbral a partir del cual se considera la existencia del contorno, de este modo los píxeles que presentan un gradiente mayor que el umbral se catalogan como contornos.

2.4. Medidas de similitud

Existen numerosas aplicaciones en el campo de procesamiento de imagen que utilizan medidas de similitud para diferentes propósitos. En algunos casos, el objetivo es la medición en si misma de la imagen, pero en otras ocasiones se utiliza con el objetivo de evaluar el rendimiento del procesamiento de la imagen. El proceso común para evaluar dicho rendimiento es el siguiente: se toma la imagen original y se daña artificialmente con un cierto ruido, entonces se procesa la imagen y se compara la procesada con la imagen original, esto permite ajustar los parámetros del filtro para conseguir un rendimiento óptimo.

Con la finalidad de comprobar la eficiencia de los filtros utilizados, se proponen diferentes métricas que midan la semejanza entre imágenes, de este modo podremos ver de modo cuantitativo si nuestro filtro mejora o no. El objetivo de la restauración es obtener una imagen lo más cercana posible a la imagen original sin ruido, por tanto estas medidas compararan la similitud entre la imagen filtrada y la imagen original libre de ruido.

Durante los últimos años, numerosos trabajos han abordado el problema de la definición de métricas que coincidan con la percepción humana, ejemplo de esto es WSNR (weighted signal to noise ratio), [20], que simula el sistema de filtrado humano. Otras medidas evalúan las diferencias en el dominio de la frecuencia y los cambios de los bordes.

Veremos en primer lugar las medidas clásicas y más utilizadas como es el PSNR, [8], [2], y tras ello una medida más actual y mejor adaptada a la percepción humana denominada FCSS, [18].

Valor pico de la relación señal ruido (PSNR)

El valor pico de la relación señal ruido (PSNR), es un parámetro usualmente utilizado en procesamiento de imágenes y es una medida relativa de la calidad de la imagen. El PSNR calcula el parecido entre dos imágenes I e $\widehat{I}$, cuyos píxeles tienen valores de intensidad comprendidos entre 0 y 255, produciendo un resultado en decibeles (dB), el cual nos da una idea del grado de similitud entre ambas imágenes.

Definición 2.4.1. *Se denomina error cuadrático medio (Mean Square Error, MSE) para dos imágenes, I e $\widehat{I}$ de tamaño $m \times n$ como:*

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \left[I(i,j) - \widehat{I}(i,j) \right]^2$$

donde $I(i,j)$ e $\widehat{I}(i,j)$ representan un píxel de la imagen original I y la imagen procesada $\widehat{I}$, respectivamente.

Con esto, definimos el PSNR como:

$$PSNR = 10 \log_{10} \frac{255^2}{MSE}$$

Un valor bajo de MSE, se traduce como menos error en la imagen procesada con respecto la imagen original, y por tanto un valor grande de PSNR indica que la relación señal ruido es grande, y por tanto buena.

Error Absoluto medio (MAE)

El error absoluto medio (MAE, Mean Absoluto Error) es un método de medición de distorsión cuantitativa en imágenes definido, utilizando las notaciones anteriores por:

$$MAE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \left[I(i,j) - \widehat{I}(i,j) \right]$$

FCSS (Fuzzy Colour Structural Similarity)

Sea I la imagen en formato RGB y sea W la ventana elegida de tamaño finito $q \times q = n$. Denotaremos por $x_i(l)$ con $i = 1, \ldots, n$ y l los distintos canales (R, G, B).
Sea ahora $\widehat{I}$ la imagen a comparar con I.

Se medirá la similitud entre dos imágenes I e $\widehat{I}$ como el promedio de las similitudes entre las ventanas I_W e $\widehat{I}_W$ obtenido cuando se desliza la ventana a lo largo de las filas de la imágen.

La medida que se propone en [18] se basa en evaluar en ambas imágenes el contraste, la estructura y la luminosidad.

Para eso necesitaremos medir la similitud entre todos los pídeles $x_i \in I_W$ e $y_i \in \widehat{I}_W$ y el vector media en cada ventana, que denotaremos por $\overline{I_W}$, $\overline{\widehat{I}_W}$.

Denotaremos estas similitudes por M_{x_i} y M_{y_i} y mediremos utilizando la métrica fuzzy, es decir

$$M_{x_i} = M(x_i, \overline{I_W}, t) = \prod_{l=1}^{3} \frac{min(x_i(l), \overline{I_W}(l)) + t}{max(x_i(l), \overline{I_W}(l)) + t}$$

con $i = 1, \ldots, n$, $t > 0$ y

$$\overline{I_W} = \frac{1}{n} \sum_{j=1}^{n} x_j$$

Con un procedimiento análogo con la imagen Y, obtenemos M_{y_i}.

Contraste El contraste se puede ver como la mayor diferencia entre I_W e $\widehat{I}_W$ se mide como

$$C_{I_W} = \text{máx}(M_{x_i}) - \text{mín}(M_{x_i})$$
$$C_{\widehat{I}_W} = \text{máx}(M_{y_i}) - \text{mín}(M_{y_i})$$

con $i = 1, \ldots, n$, y, por tanto, la similitud fuzzy entre los contrastes será

$$SC(I_W, \widehat{I}_W) = 1 - \mid C_{I_W} C_{\widehat{I}_W} \mid$$

Estructura La estructura describe como se distribuyen espacialmente las diferencias entre los píxeles y viene dada por la expresión siguiente

$$SS(I_W, \widehat{I}_W) = \frac{\sum_{i=1}^{n} 1 - \mid M_{x_i} - M_{y_i} \mid}{n}$$

Luminosidad Para comparar la luminosidad utilizaremos coordenadas esféricas [19], luego definimos la luminosidad como

$$L_{x_i} = \sqrt{x_i(1)^2 + x_i(2)^2 + x_i(3)^2}$$

y, por tanto, la similitud de luminosidad entre I_W e $\widehat{I}_W$ viene dada por

$$SL(I_W, \widehat{I}_W) = \frac{2\overline{L_{I_W} L_{\widehat{I}_W}}}{\overline{L_{I_W}^2} + \overline{L_{\widehat{I}_W}^2}}$$

donde $\overline{L_{I_W}}$ y $\overline{L_{\widehat{I}_W}}$ son las medias de luminosidad en cada ventana.
En el caso en que $\overline{L_{I_W}} = \overline{L_{\widehat{I}_W}} = 0$ se asignará $SL(I_W, \widehat{I}_W) = 1$.

Finalmente, la similitud entre I_W e $\widehat{I}_W$ resulta de la combinación de las tres medidas anteriores del modo siquiente:

$$S(I_W, \widehat{I}_W) = SC^{\alpha} SS^{\beta} SL^{\gamma}$$

donde α, β , $\gamma > 0$ son parámetros usados para adaptar la importancia de cada una de las tres componentes.

En la práctica se ha utilizado $\alpha = \beta = \gamma = 1$ y ventanas 4×4 ($q = 4$).

2.5. Wavelets

Aunque la Transformada de Fourier ha sido el soporte principal en el procesamiento de imágenes basados en transformadas desde finales de 1950, una transformada más reciente, llamada Wavelet ha facilitado el análisis, compresión y trasmisión de las mismas. La transformada Wavelet, es una herramienta matemática desarrollada a mediados de los años ochenta. Nace de la necesidad de analizar funciones, dado que la naturaleza de muchos procesos puede ser modelados mediante funciones.

El análisis con Wavelets discretas consiste en descomponer una señal en subseñales escaladas de la wavelet original, las tendencias y las fluctuaciones. La subseñal de tendencia es prácticamente la señal original pero a una menor resolución, la subseñal de fluctuación es la que almacena la información sobre los

cambios de la señal original y nos permite detectar anomalías como puede ser el ruido, o comprar patrones para detectar formas o bordes en una imagen. Existen diversas familias de wavelets (Haar, Daubechies, Coiflets,...), la elección de estas dependerá de las caracterísitcas de la señal/ imagen que queramos tratar.

Para el tratamiento de imágenes, las wavelets discretas no dan resultados satisfactorios, lo que lleva a la utilización de otro tipo de wavelets, las wavelets continuas, que disponen de un mayor número de parámetros y por tanto permiten destacar detalles y detectar texturas desde distintos ángulos.

Las wavelets son una potente herramienta tanto para reducción de ruido, como compresión de imágenes o detección de bordes, como no trataremos con esta herramienta en el presente trabajo, para ampliar véase [5], [6].

Capítulo 3

Teoría de Grafos

En este capítulo vamos a tratar los conceptos de teoría de grafos necesarios para el posterior desarrollo del filtro sin entrar en mucho detalle, para ampliar el contenido consular [15], [23], [24].

Definición 3.0.1. *Un grafo G es un par $G = (V; E)$, donde V es el conjunto de nodos o vértices y E es el conjunto de aristas. Cada arista de $a \in E$ conecta dos vértices de V, que llamaremos extremos de la arista y denotaremos como $a = (x; y)$, diremos entonces que x e y son adyacentes por a.*

Un nodo que no está conectado a ninguna arista se le denomina nodo aislado.

Definición 3.0.2. *Un digrafo o grafo es un grafo dirigido $D = (V; E)$, donde V es el conjunto de nodos y A es el conjunto de arcos o aristas orientadas. Cada arista de $a \in$ conecta dos vértices de V, que llamaremos respectivamente extremos inicial y final del arista. Diremos entonces que x es adyacente a y, y que a incide en y.*

Definición 3.0.3. *Se dice que un grafo es completo cuando todos sus vértices son adyacentes a todos los demás vértices, es decir, si todo par de vértices son los extremos de una arista.*

Definición 3.0.4. *Un grafo se dice que es ponderado si tiene asociado una función*

$$w : E \longrightarrow \mathbb{R}$$

llamada función de ponderación.

Definición 3.0.5. *Un camino en un grafo $G = (V, E)$ es una sucesión $a = a_1 \dots a_n$ de aristas tal que existe una sucesión $p = p_0 \dots p_n$ de nodos tal que para cada a_i, $P(a_i) = (p_{i-1}, p_i)$. La sucesión p se denomina recorrido del camino a.*

Un camino se dice simple si en la sucesión de vértices no hay ninguno repetido.

Teorema 3.0.6. *Si en un grafo existe un camino que conecta dos vértices, entonces existe un camino simple con extremos dichos vértices.*

Definición 3.0.7. *Un grafo se dice que es conexo si cada par de sus vértices están conectados. Es decir, G es conexo $\Leftrightarrow \ \forall\, u, v : \ \exists\, \mu = (u; v)$. En caso contrario, diremos que G es un grafo no conexo.*

En general, si G es un grafo, podemos definir en el conjunto de vértices la relación:

$$uRv \text{ si existe un camino de } u \text{ a } v.$$

Esta relación es una relación binaria de equivalencia.

Definición 3.0.8. *Dado un grafo G, las clases de equivalencia definidas en el conjunto de sus vértices por la relación de equivalencia anterior, reciben el nombre de componentes conexas de G.*

3.1. Representación matricial de grafos

En ocasiones, sobre todo a la hora de programar los algoritmos de búsqueda, resulta útil representar los grafos mediante matrices.

Definición 3.1.1. *Sea G un grafo cuyo conjunto de vértices es $V = \{v_1, v_2, \ldots, v_n\}$. Se define su matriz de adyacencia como la matriz $A \in \mathfrak{M}_n(\mathbb{R})$ cuyo coeficiente (i, j) es igual al número de aristas e que unen v_i con v_j. En caso de tener un bucle, se pondrá un 1 en el elemento de la diagonal correspondiente.*

En caso de tener un digrafo, el coeficiente (i, j) será el número de aristas que van desde el vértice v_i al v_j.

En general consideraremos grafos que no sean multigrafos, es decir, si dos vértices están conectados sólo lo estarán por una única arista.

Proposición 3.1.2. *Sea G un grafo cuyo conjunto de vértices es $\{v_1, v_2, \ldots, v_n\}$ y sea A su matriz de adyacencia. Entonces el coeficiente (i, j) de la matriz A^n es igual al número de caminos de longitud n que unen v_i con v_j.*

Corolario 3.1.3 (Caracterización de un grafo conexo)**.** *Sea A la matriz de adycencia del grafo G de vértices $\{v_1, v_2, \ldots, v_n\}$ con $n > 1$ y sea $C = A^{p-1} + A^{p-2} + \ldots + A$. El grafo G es conexo si, y sólo si todos los elementos de la matriz C son distintos de cero.*

Definición 3.1.4. *Sea G un grafo cuyos vértices y aristas son, respectivamente, $V = \{v_1, v_2, \ldots, v_n\}$ y $E = \{a_1, a_2, \ldots, a_p\}$. Llamaremos matriz de incidencia del grafo G a la matriz $M = (m_{ij})$ de modo que $m_{ij} = 1$ si v_i y a_i son incidentes, (0 en otro caso).*

En digrafos:

$$m_{ij} = \begin{cases} 1 & \text{si } v_i \text{ es el extremo origen de la arista } a_j \\ -1 & \text{si } v_i \text{ es el extremo final de la arista } a_j \\ 2 & \text{si hay un bucle} \\ 0 & \text{en otro caso} \end{cases}$$

3.2. Caminos mínimos

En muchas situaciones reales las conexiones (aristas) que tienen los grafos tienen características propias, es decir, un valor. En estos grafos ponderados el interés está en encontrar el camino que permita pasar de un nodo a otro con el menor valor (de tal manera que la suma de los pesos de las aristas que lo formen sea mínima), es lo que se denomina encontrar el camino mínimo entre dos nodos.

Existen diversos algoritmos de búsqueda de caminos óptimos, en la mayoría una manera eficaz de representar todo el proceso se consigue mediante una tabla de $n+2$ columnas, una por cada vértice, una que indique la etapa del algoritmo y la última que marque el óptimo en cada iteración, también hay que tener en cuenta que estos caminos no tienen porqué ser únicos.

A continuación se presentan los algoritmos más usuales en función del nodo que se fije para inicializar la búsqueda.

Algoritmo de Bellman-Kalaba

Bellman estudió problemas de optimización secuenciales que se componían en etapas y estableció el principio de Bellman que dice *"Todo procedimiento óptimo sólo puede estar formado por sub-procedimientos óptimos"*. El algoritmo de Bellman-Kalaba calcula caminos mínimos entre cualquier nodo y un **vértice final fijado**.

Dado un grafo $G = (V, E)$ con n vértices donde $c(x, y)$ es el valor de la arista (x, y), se fija el vértice final y se define la función de retorno optimal $V_j(x)$, como la función que proporciona el coste mínimo para alcanzar el vértice final con k aristas o menos desde el vértice x.

1. $k = 0$, se calcula para todo vértice x,
$$V_0(x) = 0 \text{ si } x \text{ es el vértice final.}$$
$$V_0(x) = \infty \text{ en otro caso.}$$

2. $k = k + 1$, $\forall\, x \in V$,
$$V_k(x) = \text{mín}\{\min_{\forall (x,y)\in A, x\neq y}\{c(x, y) + V_{k-1}(y)\}, V_{k-1}(x)\}$$

3. Se repite el Paso 2, hasta que $V_k(x) = V_{k-1}(x) = V(x)$ para todo nodo x, donde $V(x)$ es el coste, longitud o peso mínimo del camino entre el nodo x y el vértice final fijado.

Se reconstruyen los caminos óptimos teniendo en cuenta que, $V(x) = c(x, y) + V(y)$

Coste computacional del algoritmo: $\Theta(\mid E \mid\mid V \mid)$, con V el conjunto de vértices y E el conjunto de aristas.

Algoritmo de Dijkstra

Suponiendo los pesos entre los pares de vértices no negativos, el algoritmo de Dijkstra calcula los caminos mínimos entre un nodo inicial fijado y todos los demás nodos. El proceso considera etiquetas temporales para cada vértice que representan cotas superiores de las distancias mínimas del vértice origen al resto y que se van convirtiendo en permanentes en cada iteración.

Sea V el conjunto de vértices, $V = P \cup T, (P \cap T = \emptyset)$ donde P es el conjunto de nodos permanentes, T el conjunto de nodos transitorios y $x = 1$ el nodo origen (si no es así, se pueden renumerar los vértices).

Se define $V_k(j)$ como la longitud mínima del camino que une el nodo origen con el vértice j en la etapa k, donde $c(i, j)$ es la longitud de la arista (i, j).

1. $P = 1, T = P - 1, k = 0, V_0(1) = 0$, para todo $j \in T$.

   $$V_0(j) = c(1, j) \text{ si existe la arista } (1, j).$$
   $$V_0(j) = \infty \text{ si no existe la arista.}$$

   Sea h_k el nodo que verifica $V_0(h_k) = \text{mín}_{\forall j \in T}\{V_0(j)\}$ para todo vértice j transitorio.

2. $P = P \cup \{h_k\}, T = T - \{h_k\}, k = k+1$ Para todo vértice j transitorio$(j \in T)$:

   $$V_k(j) = \text{mín}\{V_{k-1}(j), V_{k-1}(h_{k-1}) + c(h_{k-1})\}$$

siendo h_{k-1} el último nodo que ha encontrado como permanente.
Sea h_k el nodo que verifica $V_k(h_k) = \text{mín}\{V_k(j)\}$ para todo vértice j transitorio.

3. Repetir el Paso 2 hasta que los nodos sean permanentes, $T = \emptyset$.

Para reconstruir el camino óptimo entre el nodo origen y el resto de vértices se va marcando la secuencia de incorporaciones al conjunto de nodos permanentes.

Coste computacional del algoritmo: $\Theta(\mid V \mid^2)$, con V el conjunto de vértices.

Algoritmo de Floyd

Permite calcular los caminos mínimos entre cualquier par de nodos sin necesidad de fijar el nodo origen o el nodo final. Sea C la matriz de pesos, tiempos, distancias o costes, es decir el elemento c_{ij} es el coste o peso de la arista (i, j), si no existe dicha arista el valor será infinito.

Sea D la matriz de penúltimos vértices, es decir, es una matriz cuadrada cuyo elemento d_ij es el penúltimo vértice del camino que une el nodo i con el j. Inicialmente se parte de las aristas que forman el grafo, por lo que en la etapa $k = 0$, $d_{ij} = i$.

1. $k = 0$, se forman las matrices C y D iniciales.

2. $k = k + 1$, se selecciona la fila y la columna k y para todo h, i distinto de k se evalúa:
$$\text{Si } c_{hk} + c_{ki} < c_{hi} \Rightarrow c_{hi} = c_{hk} + c_{ki}, \text{ y } d_{hi} = d_{ki}$$

3. Se repite el Paso 2, hasta completar todas las filas y columnas.

La matriz C resultante contiene los valores óptimos de los caminos mínimos entre cualquier par de nodos y los elementos de la matriz D son los penúltimos vértices de dichos caminos óptimos.

Coste computacional del algoritmo: $\Theta(\mid V \mid^3)$, con V el conjunto de vértices.

3.3. Árboles

Estudiaremos ahora un tipo especial de grafos, los llamados árboles. Este tipo de grafos fueron estudiados por primera vez por Kirchhoff, en 1847, en su trabajo de redes eléctricas. Sin embargo, estas estructuras son hoy en día muy importante para el estudio de diferentes problemas.

Definición 3.3.1. *Un grafo G se denomina árbol si es conexo y no contiene ciclos. En general, un grafo que no contiene ciclos se denomina bosque y cada componente conexa de un bosque es un árbol.*

Propiedades

- Dados dos vértices cualesquiera, existe un único camino simple entre ellos.
- Al eliminar una arista cualquiera del grafo, se obtiene un grafo con dos componentes conexas.
- Si el grafo tiene n nodos, el número de aristas es $m = n - 1$.
- Si a un árbol se le añade una arista más, se genera un ciclo.

Definición 3.3.2. *Llamaremos árbol generador (o árbol de expansión) de un grafo conexo a cualquier subgrafo con el mismo número de vértices que sea árbol.*

En grafos ponderados se llama árbol minimal o de peso mínimo a un árbol cuya longitud (suma del valor de sus aristas) es mínima. Se llama árbol maximal o de peso máximo a un árbol cuya longitud es máxima.

Caracterización de árboles: Un grafo no dirigido sin lazos es un árbol si y solo si todo par de vértices está unido por un único camino.

A continuación veremos algunos de los algoritmos que nos permiten obtener árboles minimales (o maximales):

Algoritmo de Kruskal

Calcula árboles mínimos sobre el grafo (si lo que se desea son los maximales basta con cambiar el valor mínimo por el máximo en cada paso).

1. Se toma el arco o arista de menor valor y se señala (o se marca). Si hubiera más de uno se toma cualquiera de ellos.

2. Se elige entre las aristas (o arcos) no marcadas aquella de menor valor que tenga como extremo cualquier nodo con alguna arista (o arco) marcada, sin dar lugar a ciclos.

3. Se repite el Paso 2 hasta conseguir tener todos los nodos conectados (en ese momento si se marca una nueva arista se formarán ciclos).

Las aristas marcadas determinan el árbol minimal.

Coste computacional del algoritmo: $\Theta(\mid E \mid \log \mid V \mid)$, siendo E el conjunto de aritas y V el conjuto de vértices.

Algoritmo de Solin

Calcula el árbol minimal o maximal a partir de la matriz de costes, $C = (c_{ij})$ donde c_{ij} es el valor de la arista (o arco) que une el nodo i con el vértice j, (si no existe la arista se puede poner infinito en el caso de árbol mínimo o no poner nada).
Para árbol mínimo:

1. Se selecciona el valor mínimo de la matriz de costes. Si hubiera varios se elige cualquiera.

2. Se elimina o se tacha la columna que posea el valor seleccionado y se marca o selecciona la fila correspondiente a la columna eliminada. De entre las filas marcadas se selecciona el valor mínimo entre los que no están tachados.

3. Se repite el Paso 2 hasta tener todas las filas marcadas y las columnas tachadas.

Los valores seleccionados proporcionan el árbol mínimo. Los dos primeros valores seleccionados pertenecen a la misma arista, por lo que sólo uno debe tenerse en cuenta a la hora de calcular la longitud del árbol.

Coste computacional del algoritmo: $\Theta(\mid E \mid \log \mid V \mid)$, siendo E el conjunto de aritas y V el conjuto de vértices.

Capítulo 4

Métodos desarrollados y resultados

4.1. Teoría de grafos en imágenes

La teoría de grafos para el tratamiento de imágenes ha sido ya utilizada previamente por muchos autores, [16], [15].

Existen numerosas posibilidades de establecer una función de una imagen en un grafo, la más obvia, es asignar a cada vértice del grafo un píxel de la imágen, de manera que el valor del vértice sea el nivel de intensidad del píxel. De este modo establecemos una correspondencia biyectiva:

$$(x, y) \longleftrightarrow i$$

siendo (x, y) un punto de la imagen, con una intensidad que denotaremos por $I(x, y)$ e i el correspondiente vértice del grafo, que denotaremos por v_i.

El peso de la arista que une el vértice i y el j, $p_{i,j}$ lo tomaremos como la distancia de un vértice a otro, pudiendo utilizar distintas métricas para ello, la más sencilla, en caso de imágenes en escala de grises se utilizará:

$$p_{i,j} = d(v_i, v_j) =\mid v_i - v_j \mid$$

y en caso de imágenes a color se utilizará la distancia euclídea.

Como ya hemos dicho anteriormente, un problema fundamental del análisis de imágenes es la eliminación de ruido y la partición de una imagen en regiones, es decir, identificar las partes más homogéneas y las partes de bordes.

Las primeras ideas para eliminar el ruido están basadas en métodos lineales, como AMF (Arithmetic Mean Filter) [21]. Este filtro elimina bien el ruido, sin embargo, difumina los bordes de manera considerable. Por esta razón se comienza con el estudio de métodos no lineales, para intentar superar este inconveniente.

Figura 4.1: Imagen filtrada con AMF

Los recientes métodos no lineales mejoran el rendimiento con respecto a los lineales, aún más en zonas de borde. Sin embargo, si el ruido de la imagen es grande, en ocasiones el ruido en regiones homogéneas se confunde con estructuras de la imagen (bordes) que deben ser preservadas y no se reduce el ruido.

En [13] se presenta un filtro que pretende vencer este inconveniente, que será el precedente de lo que presentaremos.

El trabajo propone un método que clasifica los píxeles de una foto en

- Regiones homogéneas
- Regiones de borde

con motivo de aplicar dos filtros diferentes según la región en la que nos encontremos, un filtro con mucha capacidad de suavizado para zonas homogéneas y otro

filtro no lineal para el resto de la imagen. De este modo se crea un filtro adaptativo en el que desaparece el inconveniente de confundir el ruido con zonas de detalles cuando el ruido es alto.

Un método para realizar esta segmentación se basa en la teoría de grafos. El objetivo de la segmentación es agrupar los píxeles que sean similiares en algún sentido y separar los que sean distintos respecto a la misma medida. La manera más sencilla, separar utilizando la distancia d que acabamos de definir.

En el grafo, los píxeles más semejantes, estarán unidos por las aristas de menor peso, por tanto tiene sentido aplicar los algoritmos vistos en el capítulo anterior para calcular los caminos mínimos de grafo, y así conocer que píxeles guardan mayor parecido. Existen otras maneras de segmentar una imagen utilizando teoría de grafos, pero nos centraremos en los algoritmos vistos, y en concreto, utilizaremos el algoritmo de Kruskal.

En dicho trabajo se desarrolla un método de procesamiento de una imagen I, por medio de un enfoque de "ventana deslizante". Cada píxel es procesado utilizando sus píxeles vecinos en una ventana $n \times n$, W centrada en x_i. Se definen los siguientes coeficientes:

$$c_1 = \frac{\log w(K_m)}{w(K_M)} \quad c_2 = \frac{w(K_m) - w(K_M)^2}{w(G)}$$

donde K_M y K_m son los arbóles de máximo peso y mínimo peso respectivamente del grafo G, calculados con el algoritmo de Kruskal y w denota el peso, de modo que

$$w(a_{ij}) =\mid x_i - x_j \mid$$

siendo a_{ij} la arista que une los píxeles x_i y x_j.

La razón de estos coeficientes es que sus valores son muy diferentes en zonas homogéneas y en zonas de borde cuando la imagen es ruidosa. Los valores altos de c_1 y c_2 estan asociados a regiones homogéneas, mientras que los valores bajos de los mismos indican zonas de detalles.

El filtro propuesto, SSGD (Soft-switching filtering structure) utiliza el filtro AMF (Arithmetic Mean Filter) para zonas homogéneas de la imagen y para zonas de borde un método llamado FNRM (Fuzzy Noise Reduction Method) [14]. De este modo se construirá el nuevo filtro como una combinación lineal convexa

de estos dos filtros:

$$SSGD = \alpha AMF + (1 - \alpha)FNRM$$

El valor de α se tomará en función de los valores c_1 y c_2 que hemos definido, y denotaremos por SSGD C_1F cuando se utilize c_1 y SSGD C_2F cuando se utilize c_2.

FNRM

La idea general de este método, desarrollado en [14], es mejorar cada píxel utilizando el valor de sus píxeles vecinos, teniendo en cuenta las estructuras de la imagen que deben conservarse. Para ello se utilizan 3 $2-D$ distancias de cada píxel, es decir, se calcularán las distancias entre canales dos a dos (rojo-verde, rojo-azul, verde-azul), y consideraremos cada píxel como un vector.

Denotaremos las componentes roja, verde y azul de un cierto píxel de una imagen ruidosa por:

$$N(x, y, 1) \quad N(x, y, 2) \quad N(x, y, 3)$$

respectivamente. Luego, para cada píxel, tenemos 3 componentes que definen el color.

Para cada (x, y) definimos también las siguientes parejas:

$$\begin{aligned} rg(x, y) &= (N(x, y, 1), N(x, y, 2)) \\ rb(x, y) &= (N(x, y, 1), N(x, y, 3)) \\ gb(x, y) &= (N(x, y, 2), N(x, y, 3)) \end{aligned}$$

Para cada píxel (x, y) se utilizará un ventana de tamaño $(2K + 1) \times (2K + 1)$ centrada en (x, y). Después se asignará a cada píxel de la ventana Ω unos determinados valores

$$\omega(x + k, y + l, n)$$

donde $k, l \in \{-K, \ldots, K\}$, $n = 1, 2, 3$, que denotan el peso de la componente n en la posición $(i + k, j + l)$. Estos pesos se asignarán según las siguientes reglas fuzzy:

1. *Regla 1 :* Si la distancia entre $rg(i,j)$ y $rg(x+k,y+l)$ es pequeña, y también lo es la distancia entre $rb(x,y)$ y $rb(x+k,y+l)$, entonces $\omega(x+k,y+l,1)$ será grande.

2. *Regla 2 :* Si la distancia entre $rg(i,j)$ y $rg(x+k,y+l)$ es pequeña, y también lo es la distancia entre $gb(x,y)$ y $gb(x+k,y+l)$, entonces $\omega(x+k,y+l,2)$ será grande.

3. *Regla 3 :* Si la distancia entre $rb(i,j)$ y $rb(x+k,y+l)$ es pequeña, y también lo es la distancia entre $gb(x,y)$ y $gb(x+k,y+l)$, entonces $\omega(x+k,y+l,3)$ será grande.

La idea de estas reglas fuzzy es asignar pesos grandes a las vecindades que tengan colores parecidos en el centro.

La distancia entre dos parejas se calculará de acuerdo a la distancia de Minkowski,

$$D(rg(x,y),rg(x+k,y+l))$$

que viene dada por

$$\sqrt[\delta]{(N(x+k,y+l,1)-N(x,y,1))^{\delta}+(N(x+k,y+l,2)-N(x,y,2))^{\delta}}$$

generalmente, se utiliza la distancia Euclídea ($\delta = 2$), aunque podrían utilizarse otras como la distancia de Hamming ($\delta = 1$).

Para calcular el valor que expresa que la distancia entre dos parejas es pequeña, haremos uso de conjuntos fuzzy, que suelen ser representados por funciones fuzzy

$$\mu_S(x) = \begin{cases} \left(\dfrac{p-x}{p}\right)^2 & si \quad x \leq p \\ \\ 0 & si \quad x > p \end{cases}$$

con $p \in (0, \sqrt{2}]$, para la imagen normalizada N.
Se definirán 3 conjuntos fuzzy, uno por cada pareja (rojo-verde, rojo-azul, verde-azul). Todos ellos dependerán de los parámetros p_{rg}, p_{rb}, p_{gb} respectivamente, de-

finidos por:

$$p_{rg}(x,y) = \max_{k,l\in\Omega}(\gamma_{rg}(x,y,k,l))$$
$$p_{rb}(x,y) = \max_{k,l\in\Omega}(\gamma_{rb}(x,y,k,l))$$
$$p_{gb}(x,y) = \max_{k,l\in\Omega}(\gamma_{gb}(x,y,k,l))$$

siendo:

$$\gamma_{rg}(x,y,k,l) = D(rg(x,y), rg(x+k,y+l))$$
$$\gamma_{rb}(x,y,k,l) = D(rb(x,y), rb(x+k,y+l))$$
$$\gamma_{bg}(x,y,k,l) = D(gb(x,y), gb(x+k,y+l))$$

De este modo se define el output del filtro (el valor por el que reemplazaremos cada píxel) por

$$F(x,y,n) = \frac{\sum_{k=-K}^{K}\sum_{l=-K}^{K}\omega(x+k,y+l,n)\cdot N(x+k,y+l,n)}{\sum_{k=-K}^{K}\sum_{l=-K}^{K}\omega(x+k,y+l,n)}$$

con $n = 1,2,3$, y siendo

$$\omega(x+k,y+l,1) = \mu_{S_1}(\gamma_{rg}(x,y,k,l))\cdot\mu_{S_2}(\gamma_{rb}(x,y,k,l))$$

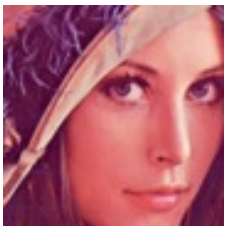
de manera análoga se define $\omega(x+k,y+l,2)$ y $\omega(x+k,y+l,3)$

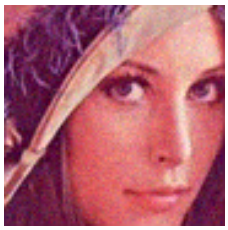
4.2. Resultados anteriores

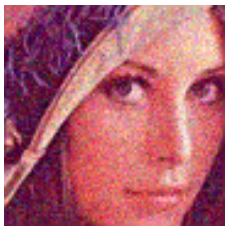
Veamos, antes de plantear el nuevo filtro, los resultados de aplicar los 3 filtros anteriores.

A continuación se presentan las imágenes que tomaremos como test de los filtros, que son las imágenes clásicas utilizadas en el procesamiento de imágenes. Se tomará la imagen inicial para poder comparar la imagen filtrada y conocer el

rendimiento del filtro, y tres imágenes a las que hemos añadido ruido Gaussiano con diferentes desviaciones estándar, σ (10, 20 y 30).


(a) Lenna

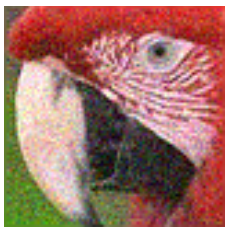

(b) Ruido $\sigma = 10$


(c) Ruido $\sigma = 20$

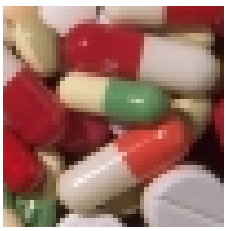

(d) Ruido $\sigma = 30$


(e) Parrots


(f) Ruido $\sigma = 10$


(g) Ruido $\sigma = 20$


(h) Ruido $\sigma = 30$

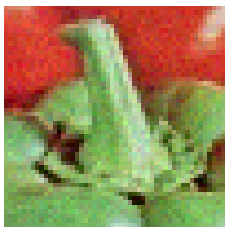

(i) Pills


(j) Ruido $\sigma = 10$

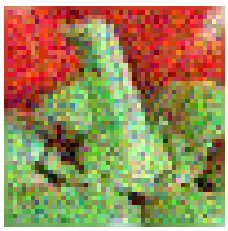

(k) Ruido $\sigma = 20$


(l) Ruido $\sigma = 30$


(m) Peppers

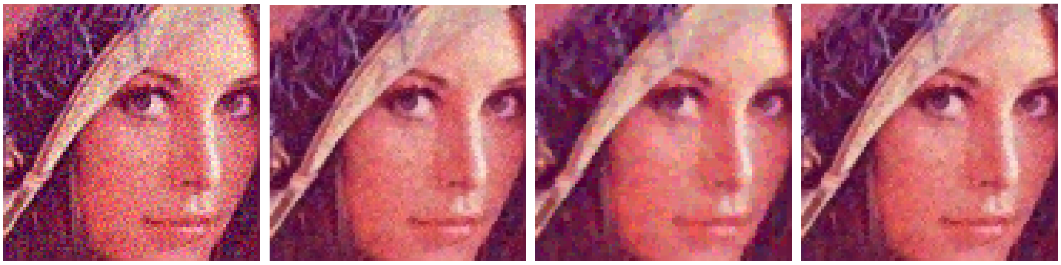

(n) Ruido $\sigma = 10$


(ñ) Ruido $\sigma = 20$


(o) Ruido $\sigma = 30$

Filtrado

Veamos visualmente el resultados de los filtros:

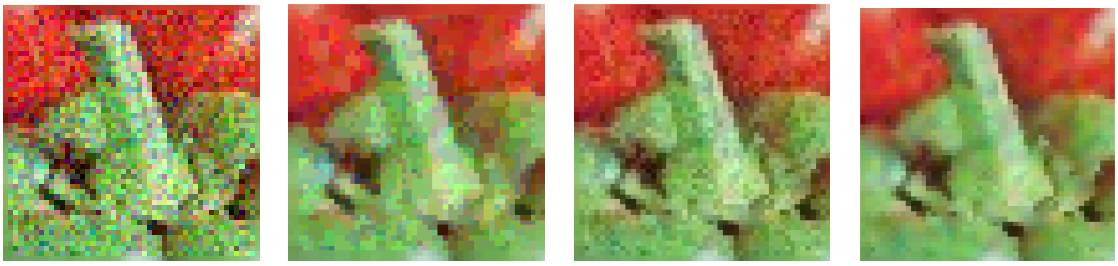


(p) (q) (r) (s)

Figura 4.2: *Filtrado Lenna con ruido Gaussiano ($\sigma = 20$)*
(a) Imagen original con ruido
(b) Imagen filtrada con FNRM
(c) Imagen filtrada con AMF
(d) Imagen filtrada con SSGD

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

Figura 4.3: *Filtrado Pills con ruido Gaussiano ($\sigma = 20$)*
(a) Imagen original con ruido
(b) Imagen filtrada con AMF
(c) Imagen filtrada con FNRM
(d) Imagen filtrada con SSGD



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

Figura 4.4: *Filtrado Peppers con ruido Gaussiano ($\sigma = 30$)*
(a) Imagen original con ruido
(b) Imagen filtrada con AMF
(c) Imagen filtrada con FNRM
(d) Imagen filtrada con SSGD

Resultados numéricos

A continuación se muestran las comparaciones en términos de PSNR, y posteriormente del FCSS utilizando las imágenes anteriores y los tres filtros expuestos, además del valor de la imagen sin filtrado:

	Parrots			**Lenna**		
σ	10	20	30	10	20	30
none	19,16	22,44	19,16	19,17	22,54	19,17
AMF	23,36	22,91	22,33	27,69	26,61	25,28
FNRM	30,84	27,36	24,66	32,53	28,14	25,20
SSGD C_1F	31,05	28,04	25,51	33,03	29,13	26,54
SSGD C_2F	31,04	28,02	25,31	33,03	29,21	26,59

Cuadro 4.1: PSNR

	Pills			**Peppers**		
σ	10	20	30	10	20	30
none	28,40	22,59	19,21	28,55	22,48	19,16
AMF	25,90	25,21	24,14	26,50	25,62	24,61
FNRM	32,28	28,14	25,13	32,14	27,84	25,16
SSGD C_1F	32,58	28,89	26,14	32,55	28,73	26,37
SSGD C_2F	32,62	28,64	26,18	32,54	28,22	26,41

Cuadro 4.2: PSNR

	Parrots			**Lenna**		
σ	10	20	30	10	20	30
none	0,9345	0,8684	0,8050	0,9336	0,8632	0,7952
AMF	0,8806	0,8624	0,8428	0,9186	0,8973	0,8745
FNRM	0,9309	0,9048	0.8756	0,9468	0,9152	0,8824
SSGD C_1F	0,9324	0,9192	0,8993	0,9509	0,9317	0,9154
SSGD C_2F	0,932	0,9187	0,8955	0,951	0,9329	0,9147

Cuadro 4.3: FCSS

	Pills			**Peppers**		
σ	10	20	30	10	20	30
none	0,9388	0.8811	0.8267	0,9353	0,8658	0,8030
AMF	0,9056	0,8843	0,8629	0,9075	0,8831	0,8610
FNRM	0,9445	0,9193	0,8912	0,9430	0,9129	0,8808
SSGD C_1F	0,9442	0,9253	0,9081	0,9474	0,9273	0,9096
SSGD C_2F	0,945	0,9252	0,9081	0,947	0,919	0,9101

Cuadro 4.4: FCSS

4.3. Métodos desarrollados

El nuevo filtro que se propone se utilizará en imágenes digitales en RGB. Esto implica que se considerará cada píxel como un vector de tres componentes, una por cada canal (Rojo, Verde y Azul).

Un primer objetivo es la segmentación de la imagen, es decir, agrupar los píxeles que sean similares y poder discernir de alguna manera cuantitativa cuando estamos ante una zona homogénea y cuando en una de bordes. Es primordial esta etapa de procesado puesto que la finalidad no sólo es dividir la imagen en zonas para un objetivo concreto, sino también utilizarlo como una primera etapa de filtrado. Conocer qué zonas son borde y qué zonas son homogéneas nos permite crear filtros adaptativos, que filtraran de un modo u otro según la zona en la que estemos.

Tomaremos para cada píxel una ventana 3×3, W, centrada en dicho píxel y construiremos un grafo como explicábamos anteriormente. Cada píxel será un nodo del grafo y las aristas unirán todos los píxeles adyacentes, cuyo peso será la distancia euclídea entre los píxeles que conecte.

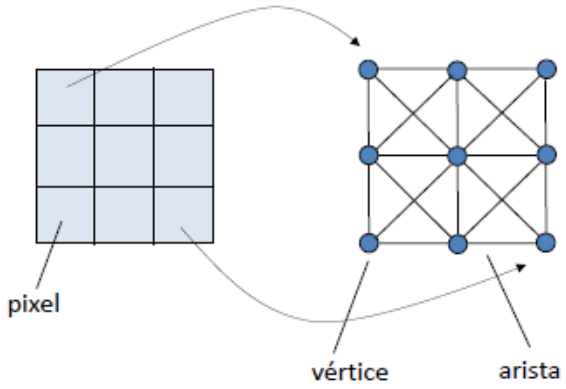


Figura 4.5: Asignación imagen-grafo

Dado que el objetivo es agrupar los píxeles que sean similares, respecto a la medida tomada (distancia euclídea), aplicaremos al grafo asociado a la ventana W el algoritmo de Kruskal para calcular el árbol mínimo.

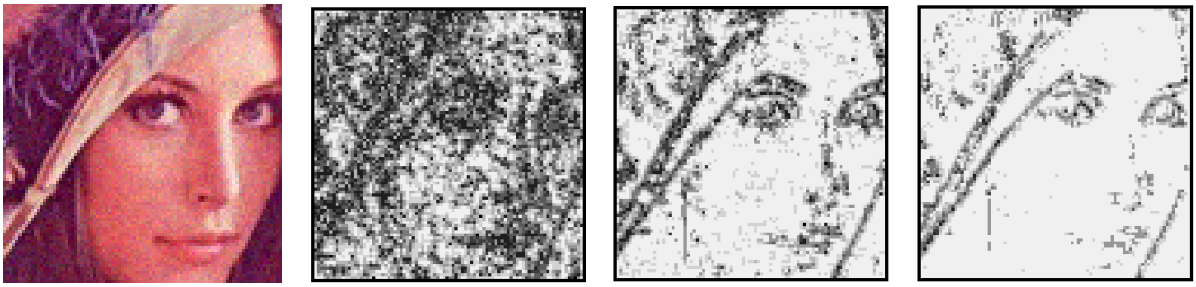
Una arista de poco peso une dos píxeles similares, mientras que las aristas de pesos grandes unen píxeles dispares.

Como vimos en el capítulo anterior, el algoritmo de Kruskal finalizará cuando todos los nodos estén conectados, esto significará, que la imagen puede presentar dos zonas bien diferenciadas y, sin embargo, nuestro algoritmo lo entenderá como una única zona.

Este inconveniente se puede vencer añadiendo un nuevo parámetro, que supondrá un nuevo algoritmo siguiendo la idea de Kruskal, al que denominaremos *umbral*. Ahora, el algoritmo finalizará cuando una arista supere el valor de dicho umbral, o bien, cuando se conecten todos los vértices, de modo que si se conectarán todos, o al menos una cantidad razonable, estaríamos ante una zona homogénea de la imagen y en caso contrario, ante una zona de borde. Este hecho nos ha permitido, tras haber aplicado Kruskal con cierto umbral, segmentar la imagen, utilizando para ello el cardinal de la componente conexa que contenga al píxel central. Es decir, para cada ventana 3×3 de la imagen, aplicaremos el algoritmo de Kruskal (con umbral) al grafo asociado a la ventana y contaremos el número de píxeles contenidos en la componente conexa que incluye al píxel central. De este modo, si este valor es alto sabremos que estamos ante una zona homogénea y, en caso contrario, en una de detalles.

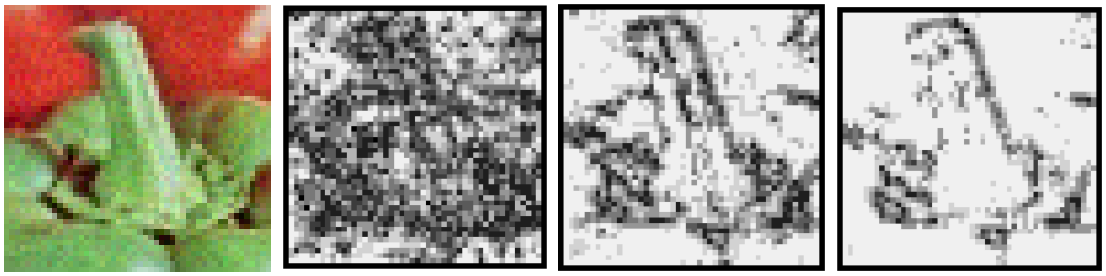
El valor de este parámetro dependerá de la imagen, tanto de lo homogénea que sea esta, como del ruido que presente.

Si construimos imágenes en tono de grises en función de este cardinal, asociando el valor de intensidad mínimo(0) al cardinal 1, y el valor de intensidad máximo (255) al cardinal 9, como se puede ver en los anexos, página 81, podemos observar qué umbral será óptimo a la hora de separar zonas de borde de zonas homogéneas:



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

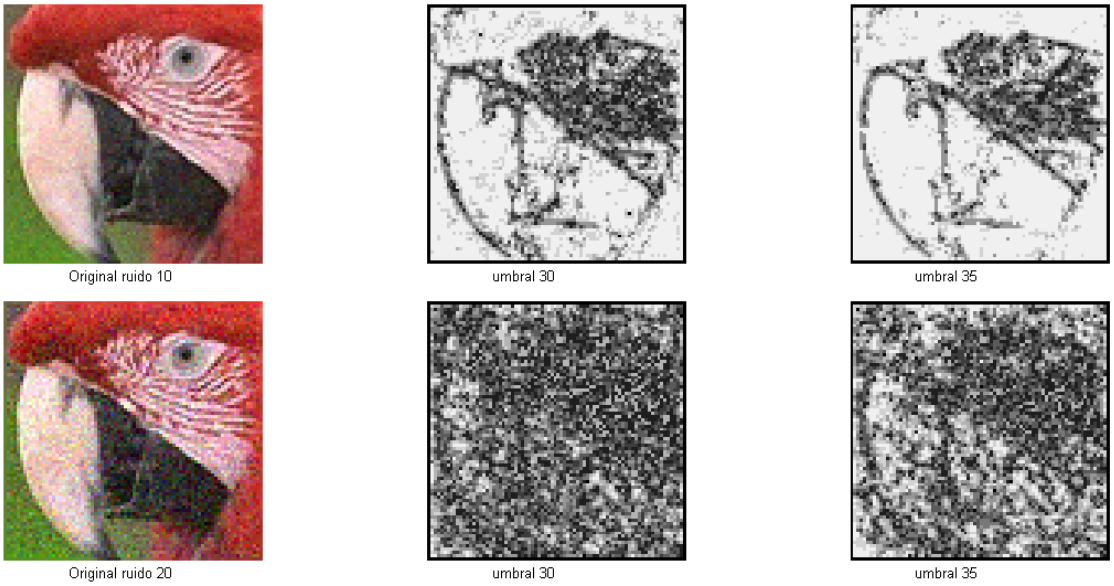
Figura 4.6: *Clasificación en región homogéna y de bordes para Lenna (ruido 10) y diferentes umbrales*
(a) Imagen original
(b) Segmentación con umbral 20
(c) Segmentación con umbral 30
(d) Segmentación con umbral 40



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

Figura 4.7: *Clasificación en región homogéna y de bordes para Peppers (ruido 10) y mismos umbrales que figura 4.6*
(a) Imagen original
(b) Segmentación con umbral 20
(c) Segmentación con umbral 30
(d) Segmentación con umbral 40

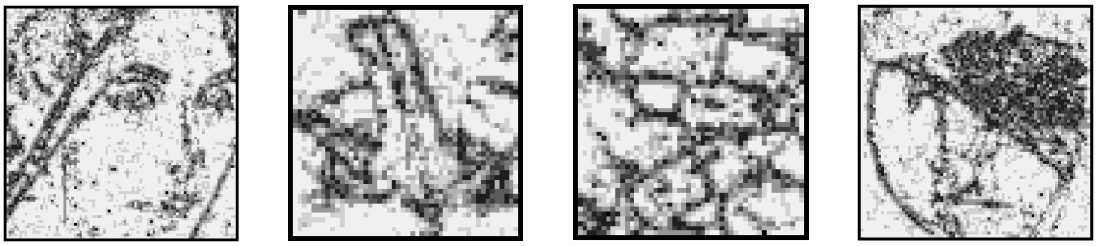
La variación de ruido alterará el valor de umbral óptimo para la segmentación; a medida que el ruido aumenta, se confunden con mayor frecuencia las zonas de detalles con dicho ruido, lo que provocará un deterioro de la segmentación.



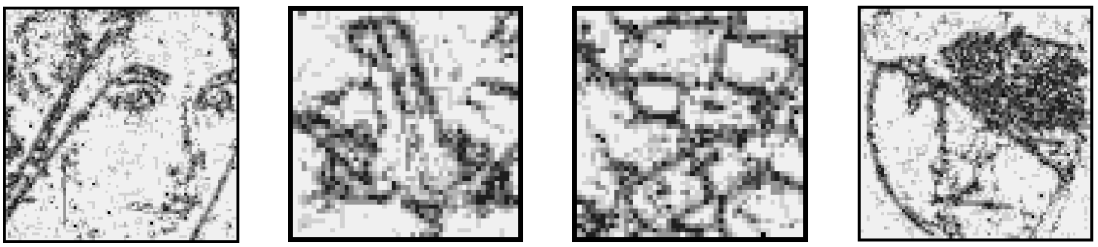
En esta figura vemos como, utilizando los mismos umbrales, obtenemos una buena segmentación de la imagen con ruido 10, sin embargo, tomando la misma imagen con ruido 20 se obtienen peores resultados. Al aparecer más ruido en la imagen debemos aumentar el umbral para impedir que se equivoquen, como ya decíamos, las zonas de detalle con el ruido.

La elección de la imágen a segmentar también podría suponer una variación del umbral óptimo debido a las diferencias de homogeneidad y suavidad entre estas. Sin embargo, al contrario que en el caso anterior, comparando los resultados con diferentes umbrales para los distintos niveles de ruido, podemos fijar un valor del umbral que permita una efectiva segmentación en cualquier imagen

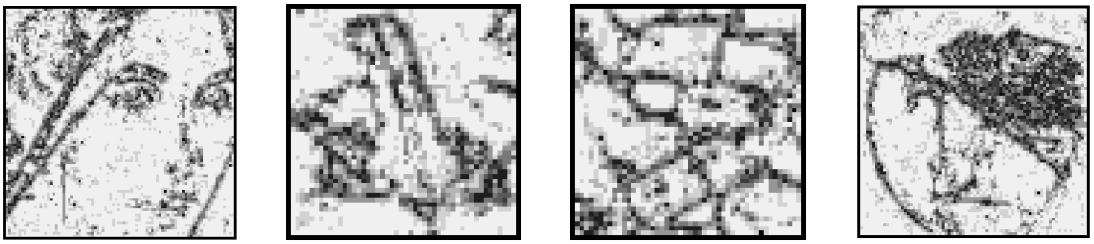
(a) Imágenes ruido 10



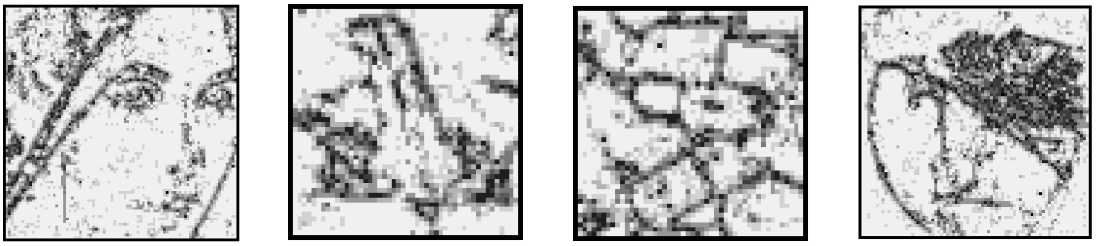
(b) Umbral 28



(c) Umbral 29



(d) Umbral 30



(e) Umbral 31

Pese a las grandes diferencias entre las imágenes tomadas para probar la segmentación, si el ruido no es muy alto, con un umbral en torno al 30 nos proporciona para todas las imágenes una buena segmentación.

4.3.1. Mejora de SSGD : SSGD-Kruskal

En la sección anterior hemos presentado un filtro que combinaba el AMF y FNRM, utilizando AMF cuando estábamos en zonas homogéneas y FNRM en zonas de borde. Para diferenciar ambas situaciones, el filtro utilizaba dos coeficientes basados en el árbol de coste mínimo y el árbol de coste máximo, y hemos visto que proporcionaba muy buenos resultados.

Ahora, acabamos de introducir un nuevo método de segmentación que también nos permite discernir si estamos en zona suave o zona de detalles, por tanto, aplicaremos la misma idea de este filtro, cambiando la decisión de utilización de AMF y FNRM.

Crearemos una combinación de estos dos filtros, basándonos precisamente en la cardinalidad de la componente conexa del píxel central, que denotaremos en lo sucesivo por C. Cuando C este cerca de 9, ponderaremos a favor del filtro AMF y, cuando este cercano a 1 a favor del FNRM.

Esta combinación vendrá caracterizada por un vector, que denotaremos por $V_p = \{v_{p_1}, \ldots, v_{p_9}\}$ de 9 componentes, donde

$$v_{p_i} = \{\text{ponderación si la componente conexa tiene i vértices}\}$$

Esto nos permitirá en caso de que la componente conexa tenga un cardinal grande, aplicar el filtro homogéneo (AMF), y en caso de cardinal pequeño aplicar FNRM.

Por tanto, para cada ventana 3×3, W, se reemplazará el píxel central por:

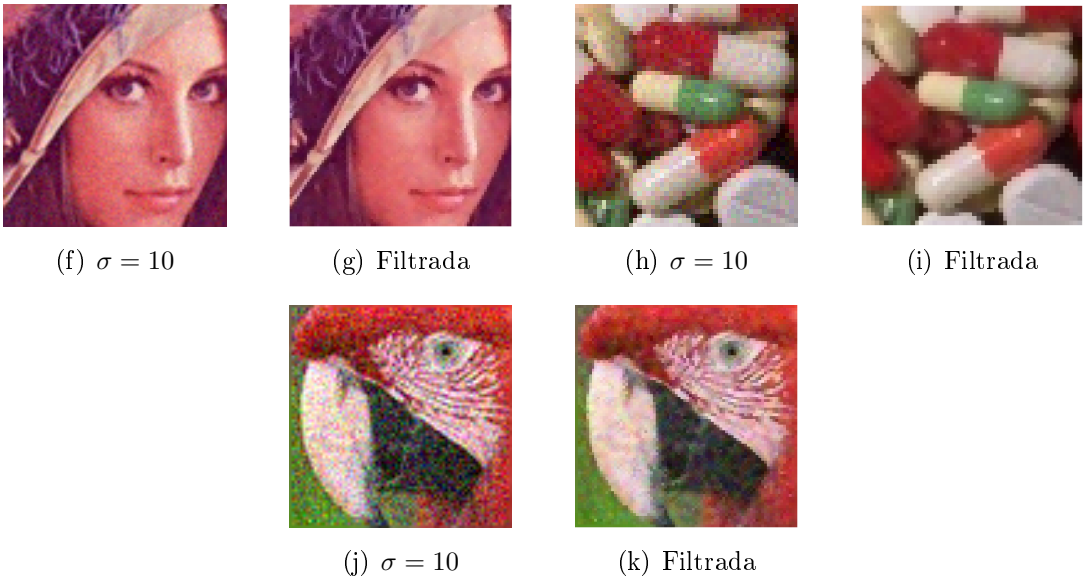
$$P_C = v_{p_\mu} AMF + (1 - v_{p_\mu}) FNRM$$

siendo $1 \leq \mu \leq 9$ la cardinalidad de la componente conexa en la que se encuentra el píxel central al hacer Kruskal con umbral.

Cuando tengamos componentes conexas pequeñas, por ejemplo $\mu = 1$ ó $\mu = 2$, estamos ante zonas de borde, por tanto querremos ponderar a favor

del FNRM, luego nuestro vector V_p habrá de tener componentes v_{p_1} y v_{p_2} bajas, sin embargo, cuando μ sea alto, como $\mu = 8$ ó $\mu = 9$, estaremos ante una zona homogénea y por tanto v_{p_8} y v_{p_9} deberán ser altos para ponderar a favor del AMF.

Nuestro nuevo filtro, al que llamaremos SSGD-Kruskal, dependerá de dos parámetros, el umbral U y el vector de ponderaciones V_p, que habrá que ajustar para cada nivel de ruido. A medida que el ruido en la imagen aumenta, el valor del umbral deberá aumentar para evitar confusión de ruido con zonas de detalle.



(f) $\sigma = 10$ (g) Filtrada (h) $\sigma = 10$ (i) Filtrada

(j) $\sigma = 10$ (k) Filtrada

Figura 4.8: Filtrado SSGD-Kruskal

En la última sección veremos los resultados que nos ofrece este cambio del filtro SSGD en términos del PSNR y FCSS, y estudiaremos que valor de los parámetros que maximicen estas medidas.

4.3.2. Nuevo Filtro

Como ya hemos comentado anteriormente, en ocasiones, los filtros utilizados para eliminar el ruido de una imagen, suavizan demasiado, provocando imágenes borrosas y distorsiones en los detalles de las mismas. Por esta razón, pa-

ra la implementación del nuevo filtro se utilizará la idea anterior, que nos permite ajustar el parámetro *umbral* para encontrar el equilibrio óptimo entre la eliminación de ruido y la preservación de los detalles.

Una vez tenemos el bosque generado a partir del algoritmo de Kruskal con umbral, cuya implementación en Matlab podemos ver en 82, tomaremos la componente conexa que contiene al píxel central, que nos asegura que los píxeles contenidos en dicha componente son parecidos a este, y la utilizaremos para remplazar el nodo central por una combinación de los mismos. La combinación que se propone, tendrá en cuenta la distancia existente entre cada uno de los píxeles y el píxel central, dando mayor peso a los píxeles más cercanos. Para realizar esto necesitamos conocer la distancia mínima del nodo central al resto de nodos, que calcularemos con el algoritmo de Dijkstra explicado en el capítulo anterior y que podemos ver implementado en la página 86.

Aunque existen otros algoritmos que nos permiten conocer las distancias mínimas entre todos los nodos de un grafo, como es el algoritmo de Floyd, se ha elegido el de Dijkstra por intereses computacionales. El algoritmo de Floyd calcula la distancia mínima entre cualquier par de nodos, sin embargo, para nuestro propósito solo necesitamos conocer la distancia mínima de todos los nodos al nodo central, que es precisamente lo que calcula el algoritmo de Dijkstra.

Sea G el grafo asociado a la ventana 3×3, que denotamos por W. Sea ahora $G = (V, E)$ el grafo que contiene al nodo central obtenido de aplicar Kruskal con umbral. Sea $\{v_1, \ldots, v_n\} \in V$, con $n \leq 9$ los vértices de la componente conexa del píxel central v_5, y sea d la distancia euclídea y $w(v_i)$ el peso del vértice v_i.

$$M_W = \frac{\sum_{i=1}^{n} \frac{1}{d(v_i, v_5)} w(v_i)}{\sum_{i=1}^{n} \frac{1}{d(v_i, v_5)}}$$

Y reemplazaremos el píxel central por la siguiente combinación lineal convexa:

$$PC(\lambda, \delta) = \begin{cases} \text{mediana}_{v_j \in W}\{v_j\} & \text{si} \quad C \leq \delta \\ \\ \lambda w(v_5) + (1 - \lambda) M_W & \text{si} \quad C > \delta \end{cases}$$

Nuestro nuevo filtro, que podemos ver implementado en la página 89, depende entonces de 3 parámetros:

1. U: umbral que determinará cuando finaliza el algoritmo de Kruskal.

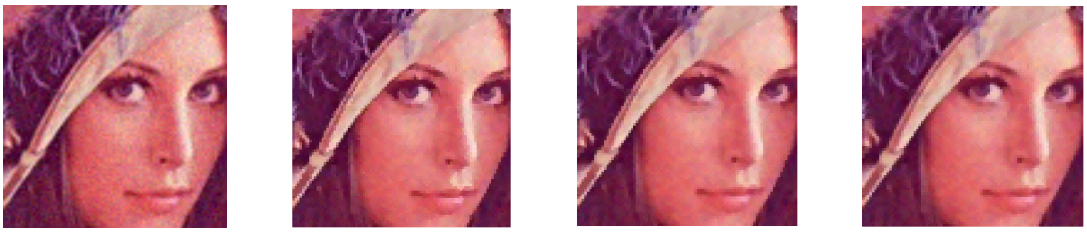
2. λ: factor de la combinación lineal convexa entre píxel central y píxeles vecinos.

3. δ: parámetro que decide si reemplazar píxel central por la combinación lineal de los píxeles de su componente conexa o por el vector mediana de su ventana W.

Veremos ahora comparaciones de imágenes variando cada uno de estos tres parámetros y así conocer el rango de valores que mejor resultado proporciona.

Dependencia del umbral U

A continuación mostramos algunas imágenes filtradas con diferentes umbrales para ver visualmente la dependencia del umbral en el proceso de filtrado:



(a) Lenna $\sigma = 10$ (b) $U = 30$ (c) $U = 50$ (d) $U = 79$

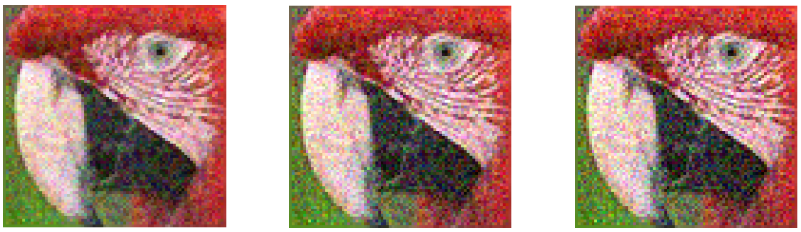
Figura 4.9: Filtrado Lenna para $\lambda = 0{,}3$ y $\delta = 1$

(a) Pills $\sigma = 20$ (b) $U = 80$ (c) $U = 100$ (d) $U = 120$

Figura 4.10: Filtrado Pills para $\lambda = 0{,}1$ y $\delta = 1$

Dependencia de λ

(a) Parrot $\sigma = 30$ (b) $\lambda = 0{,}1$ (c) $\lambda = 0{,}2$ (d) $\lambda = 0{,}3$



(e) $\lambda = 0{,}6$ (f) $\lambda = 0{,}7$ (g) $\lambda = 0{,}9$

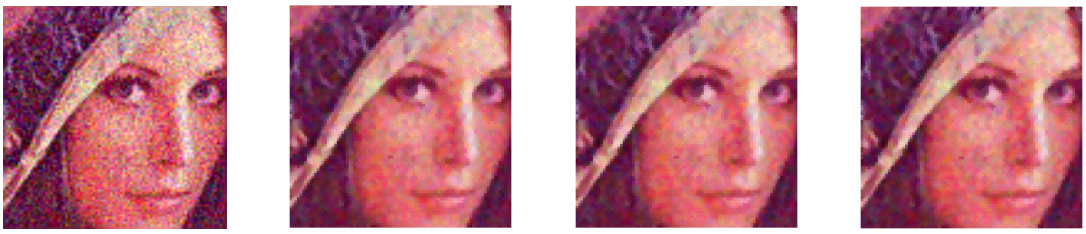
Figura 4.11: Filtrado Parrot para $U = 120$ y $\delta = 1$

Como podemos ver en las imágenes anteriores, al aumentar el valor de λ, lo que se traduce por un mayor peso al píxel central, la capacidad de eliminación de ruido disminuye y aumenta la preservación de detalles junto con el ruido, pero si es demasiado bajo en ocasiones se suaviza excesivamente. Habrá que elegir el valor de este parámetro junto con los otros dos para equilibrar la reducción de

ruido con la preservación de los detalles.

Dependencia de δ

La dependecia del parámetro δ es bastante baja puesto que la mayor parte de las componentes conexas que se generan con el nuevo procedimiento inspirado en Kruskal tendrán un cardinal alto y por tanto es fácil que al aumentar δ en valores pequeños (entre 1 y 5), el filtrado prácticamente no varíe.



(a) Lenna $\sigma = 30$ (b) $\delta = 1$ (c) $\delta = 5$ (d) $\delta = 8$

Figura 4.12: Filtrado Parrot para $U = 150$ y $\lambda = 0{,}1$

4.3.3. Resultados y comparación

Comprobamos numéricamente mediante las medidas de similitud vistas (PSNR y FCSS) el resultado de los nuevos filtros y comparamos con los tres filtros de partida: AMF, FNRM y SSGD.

Como ya hemos visto, el filtrado depende en ambos casos de parámetros, el SGGD-Kruskal depende de dos parámetros (umbral y vector de ponderaciones) y el nuevo filtro depende de tres.

Para el nuevo filtro, tras comprobar experimentalmente los valores que mejor resultado de PSNR y FCSS ofrecía, se procedió a calcular todas las combinaciones de estos tres parámetros para poder elegir el máximo de dichas medidas. Un bucle, que podemos ver en la página 91, que recorre todos los posibles valores de los parámetros dentro de un rango adecuado nos permite elegir los valores de los parámetros que maximizan el PSNR y FCSS

Aunque para cada imagen y tipo de ruido se han obtenido valores para los parámetros distintos, podemos dar un rango óptimo según el nivel de ruido:

Valores óptimos de los parámetros

Ruido	$\sigma = 10$	$\sigma = 20$	$\sigma = 30$
U	[60 , 80]	[100 , 125]	[120 , 140]
λ	[0.5 , 0.3]	[0.1 , 0.3]	0.1
δ	1	1	1

Cuadro 4.5: PSNR

Mostramos ahora el resultado proporcionado por el PSNR para los diferentes filtros expuestos con valores para los parámetros dentro de los intervalos marcados

	Parrots			**Lenna**		
σ	10	20	30	10	20	30
none	19,16	22,44	19,16	19,17	22,54	19,17
AMF	23,36	22,91	22,33	27,69	26,61	25,28
FNRM	30,84	27,36	24,66	32,53	28,14	25,20
SSGD C_1F	31,05	28,04	25,51	33,03	29,13	26,54
SSGD C_2F	31,04	28,02	25,31	33,03	29,21	26,59
SSGD Kruskal	**30,92**	**27,86**	**25,22**	**32,89**	**28,97**	**26,38**
Kruskal-Dijkstra	**30,52**	**26,40**	**23,96**	**32,029**	**28,19**	**25,78**

Cuadro 4.6: PSNR

σ	**Pills**			**Peppers**		
	10	20	30	10	20	30
none	28,40	22,59	19,21	28,55	22,48	19,16
AMF	25,90	25,21	24,14	26,50	25,62	24,61
FNRM	32,28	28,14	25,13	32,14	27,84	25,16
SSGD C_1F	32,58	28,89	26,14	32,55	28,73	26,37
SSGD C_2F	32,62	28,64	26,18	32,54	28,22	26,41
SSGD Kruskal	**32,48**	**28,62**	**26,1**	**32,40**	**28,62**	**26,20**
Kruskal-Dijkstra	**31,6**	**27,78**	**25,29**	**31,58**	**27,46**	**25,43**

Cuadro 4.7: PSNR

Tras comparar en términos tanto del PSNR, vemos que se obtienen mejores resultados, especialmente para las imágenes con ruido alto. Esto se corresponde con que el nuevo filtro desarrollado preserva de manera notable los detalles y bordes de la imagen en comparación con el FNRM, luego se ha conseguido un buen equilibrio entre suavizado y preservación de detalles.

Comparando con el filtro SSGD propuesto en [13], hemos conseguido acercarnos casi por completo a sus resultados, y, nuevamente, cabe resaltar en imágenes de ruido alto.

En cuanto al cambio de filosofía del filtro SSGD utilizando nuestro método de segmentación, se consiguen resultados satisfactorios aunque mejorables. Se ha comprobado experimentalmente que la dependencia del vector de ponderaciones V_p es muy alta, y debido al gran número de combinaciones posibles, es fácil pensar que no hemos obtenido, todavía, el máximo para este filtro.

Por último, comparamos los mismos filtros ahora en términos de FCSS, que nos da una idea cuantitativa de la similitud de las imágenes más aproximada a la percepción humana que el FNRM. Habiéndose utilizado para el filtro SSGD los siguientes valores de los parámetros:

- Para ruido con desviaciones $\sigma = 10$ y $\sigma = 20$:

$$V_p = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0{,}2 \quad 0{,}2 \quad 0{,}7 \quad 0{,}9],\ U \in [20, 40]$$

- Para ruido con desviación $\sigma = 30$:

$$V_p = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0{,}2 \quad 0{,}2 \quad 0{,}7 \quad 0{,}9],\ U \in [40, 60]$$

	Parrots			**Lenna**		
σ	10	20	30	10	20	30
none	0,9345	0,8684	0,8050	0,9336	0,8632	0,7952
AMF	0,8806	0,8624	0,8428	0,9186	0,8973	0,8745
FNRM	0,9309	0,9048	0.8756	0,9468	0,9152	0,8824
SSGD C_1F	0,9324	0,9192	0,8993	0,9509	0,9317	0,9154
SSGD C_2F	0,932	0,9187	0,8955	0,951	0,9329	0,9147
SSGD Kruskal	**0,931**	**0,9161**	**0,8933**	**0,949**	**0,9306**	**0,9102**
Kruskal-Dijkstra	**0,936**	**0,909**	**0,8783**	**0,9486**	**0,9243**	**0,9058**

Cuadro 4.8: FCSS

	Pills			**Peppers**		
σ	10	20	30	10	20	30
none	0,9388	0.8811	0.8267	0,9353	0,8658	0,8030
AMF	0,9056	0,8843	0,8629	0,9075	0,8831	0,8610
FNRM	0,9445	0,9193	0,8912	0,9430	0,9129	0,8808
SSGD C_1F	0,9442	0,9253	0,9081	0,9474	0,9273	0,9096
SSGD C_2F	0,945	0,9252	0,9081	0,947	0,919	0,9101
SSGD Kruskal	**0,9446**	**0,9228**	**0,9059**	**0,9463**	**0,925**	**0,9031**
Kruskal-Dijkstra	**0,9405**	**0,9156**	**0,8996**	**0,9410**	**0,9161**	**0,8962**

Cuadro 4.9: FCSS

Los resultados en términos del FCSS nos muestran la misma conclusión que en términos del FCSS, dado que esta medida se adapta mejor a la percepción humana, podemos afirmar que visualmente el filtrado de las imágenes con el nuevo filtro mejora el filtrado del FNRM y se acerca mucho al SSGD.

Ligeramente superiores se muestran los datos para el SSGD-Kruskal, llegando incluso en algunos casos a superar al SSDG; pero, como decíamos anteriormente, aún no se ha alcanzado el máximo de este valor y se espera que supere notablemente los filtros presentados.

Capítulo 5

Conclusiones

Se ha desarrollado un competitivo filtro de imágenes a color que elimina el ruido Gaussiano con gran efectividad, pudiéndose comparar con filtros muy utilizados actualmente como el FNRM. Además se ha modificado un filtro existente, SSGD, alcanzando los valores ya conseguidos con el mismo y dejando la puerta abierta a optimizarlo y superar a este.

Se ha comparado cualitativa y cuantitativamente ambos filtros con tres potentes filtros existentes publicados en [14], [13], llegando a resultados muy satisfactorios.

Se seguirá trabajando en el nuevo filtro, mejorando el algoritmo propuesto para alcanzar los valores máximos posibles. También se continuará estudiando condición de decisión para el filtro SSGD-Kruskal, aplicando métodos de gradiente para conseguir maximizar el PSNR y FCSS, con lo que se espera mejorar los resultados del SSGD actual.

Capítulo 6

Anexos

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Función que calcula la distancia entre los vectores V1 y V2 según    %
%            la métrica de distancia euclídea o L2-Norma               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function dist=DL2(V1,V2)

    dist = sqrt(sum((double(V1) - double(V2)).^2));

end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Función que calcula la matriz de adyacencia asociada al grafo B       %
%                   utilizando la métrica euclídea                       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [M] = MatrizAdyacenciaEuclideo(B)

    [r c d] = size(B);


    %Inicializamos matriz a infinito
    for m=1:r*c
        for n=1:r*c

            M(m,n)=inf;

        end
    end

    %Rellenamos matriz para de distancia entre nodos conectados
    %si los nodos no están conectados aparecerá infinito

    %ESQUINAS:
    %superior izquierda

        M(1,2)=DL2(B(1,1,:),B(1,2,:));
        M(2,1)=DL2(B(1,1,:),B(1,2,:));
        M(1,1+r)=DL2(B(1,1,:),B(2,1,:));
        M(5,1+r)=DL2(B(1,1,:),B(2,1,:));

    %superior derecha

        M(r,r-1)=DL2(B(1,r,:),B(1,r-1,:));
        M(r-1,r)=DL2(B(1,r,:),B(1,r-1,:));
        M(r,2*r)=DL2(B(1,r,:),B(2,r,:));
        M(2*r,r)=DL2(B(1,r,:),B(2,r,:));
```

```
%BORDES:

%fila superior

for j=2:(c-1)

    M(j,r+j) = DL2(B(1,j,:),B(2,j,:));
    M(r+j,j) = DL2(B(1,j,:),B(2,j,:));

    M(j,j+1) = DL2(B(1,j,:),B(1,j+1,:));
    M(j+1,j) = DL2(B(1,j,:),B(1,j+1,:));

%fila inferior

    M((r-1)*c+j,(r-1)*c+j-1)=DL2(B(c,j-1,:),B(c,j,:));
    M((r-1)*c+j-1,(r-1)*c+j)=DL2(B(c,j-1,:),B(c,j,:));

    M((r-1)*c+j,(r-1)*c+j+1)=DL2(B(c,j+1,:),B(c,j,:));
    M((r-1)*c+j+1,(r-1)*c+j)=DL2(B(c,j+1,:),B(c,j,:));

    %M((r-1)*r+j,(r-1)*r+j-r) = DL2(B(r,j,:),B(r-1,j,:));
    %M((r-1)*r+j-r,(r-1)*r+j) = DL2(B(r,j,:),B(r-1,j,:));

    %M((r-1)*r+j,(r-1)*r+j+1) = DL2(B(r,j,:),B(r-1,j+1,:));
    %M((r-1)*r+j+1,(r-1)*r+j) = DL2(B(r,j,:),B(r-1,j+1,:));

%columna izquierda

    M(c*(j-1)+1,c*(j-1)+1-c) = DL2(B(j,1,:),B(j-1,1,:));
    M(c*(j-1)+1-c,c*(j-1)+1) = DL2(B(j,1),B(j-1,1));

    M(c*(j-1)+1,c*(j-1)+1+c) = DL2(B(j,1,:),B(j+1,1,:));
    M(c*(j-1)+1+c,c*(j-1)+1) = DL2(B(j,1,:),B(j+1,1,:));

    M(c*(j-1)+1,c*(j-1)+1-c+1) = DL2(B(j,1,:),B(j-1,2,:));
    M(c*(j-1)+1-c+1,c*(j-1)+1) = DL2(B(j,1,:),B(j-1,2,:));
```

```
    M(c*(j-1)+1,c*(j-1)+1+c+1) = DL2(B(j,1,:),B(j+1,2,:));
    M(c*(j-1)+1+c+1,c*(j-1)+1) = DL2(B(j,1,:),B(j+1,2,:));


%columna derecha

    M(c*j,c*j-c) = DL2(B(j,c,:),B(j-1,c,:));
    M(c*j-c,c*j) = DL2(B(j,c,:),B(j-1,c,:));

    M(c*j,c*j+c) = DL2(B(j,c,:),B(j+1,c,:));
    M(c*j+c,c*j) = DL2(B(j,c,:),B(j+1,c,:));

    M(c*j,c*j-c-1) = DL2(B(j,c,:),B(j-1,c-1,:));
    M(c*j-c-1,c*j) = DL2(B(j,c,:),B(j-1,c-1,:));

    M(c*j,c*j+c-1) = DL2(B(j,c,:),B(j+1,c-1,:));
    M(c*j+c-1,c*j) = DL2(B(j,c,:),B(j+1,c-1,:));

end

for i=2:(r-1)
    for j=2:(c-1)
        M((i-1)*r+j,(i-1)*r+j-1-r)= DL2(B(i,j,:),B(i-1,j-1,:));
        M((i-1)*r+j-1-r,(i-1)*r+j)= DL2(B(i,j,:),B(i-1,j-1,:));

        M((i-1)*r+j,(i-1)*r+j-r)= DL2(B(i,j,:),B(i-1,j,:));
        M((i-1)*r+j-r,(i-1)*r+j)= DL2(B(i,j,:),B(i-1,j,:));

        M((i-1)*r+j,(i-1)*r+j+1-r)= DL2(B(i,j,:),B(i-1,j+1,:));
        M((i-1)*r+j+1-r,(i-1)*r+j)= DL2(B(i,j,:),B(i-1,j+1,:));

        M((i-1)*r+j,(i-1)*r+j-1)= DL2(B(i,j,:),B(i,j-1,:));
        M((i-1)*r+j-1,(i-1)*r+j)= DL2(B(i,j,:),B(i,j-1,:));

        M((i-1)*r+j,(i-1)*r+j+1)= DL2(B(i,j,:),B(i,j+1,:));
        M((i-1)*r+j+1,(i-1)*r+j)= DL2(B(i,j,:),B(i,j+1,:));
```

```
            M((i-1)*r+j,(i-1)*r+j-1+r)= DL2(B(i,j,:),B(i+1,j-1,:));
            M((i-1)*r+j-1+r,(i-1)*r+j)= DL2(B(i,j,:),B(i+1,j-1,:));

            M((i-1)*r+j,(i-1)*r+j+r)= DL2(B(i,j,:),B(i+1,j,:));
            M((i-1)*r+j+r,(i-1)*r+j)= DL2(B(i,j,:),B(i+1,j,:));

            M((i-1)*r+j,(i-1)*r+j+1+r)= DL2(B(i,j,:),B(i+1,j+1,:));
            M((i-1)*r+j+1+r,(i-1)*r+j)= DL2(B(i,j,:),B(i+1,j+1,:));
        end
    end

    %superior izquierda

        M(1,2)=DL2(B(1,1,:),B(1,2,:));
        M(2,1)=DL2(B(1,1,:),B(1,2,:));
        M(1,1+r)=DL2(B(1,1,:),B(2,1,:));
        M(5,1+r)=DL2(B(1,1,:),B(2,1,:));

end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%FUNCIÓN QUE NOS DA LA MEDIDA DE RAÍZ DE ERROR MEDIO CUADRÁTICO  %
%               ENTRE LAS IMÁGENES DE ENTRADA.                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function measure=RMSEImages(Image1,Image2)

    measure = sqrt(MSEImages(Image1,Image2));
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% FUNCIÓN QUE CALCULA EL VECTOR MEDIANA USANDO COMO MÉTRICA LA NORMA L2

    %Inputs:
        %Mask = mascara que incida con un 0 si un pixel no se debe tener en
                %cuenta y con cualquier otro valor en caso contrario.
                %Se debe llamar con ones() para tener todos en cuenta

        %Window = ventana 3x3

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [VM,Window0,MaxDL2,ADVM] = VM_DL2(Window,Mask)

    [a b c] = size(Window);
    Window = double(Window);
    dist = 100000*ones(a,b);
    %distancia muy alta incial para despreciar los q no incluya Mask
    VM = [0 0 0];
    MaxDL2 = 0;
    for k=1:a
        for l=1:b
            if (Mask(k,l)>0) %solo computamos distancias para píxeles buenos
                v1 = [Window(k,l,1) Window(k,l,2) Window(k,l,3)];
                %vector (k,l) de la ventana
                dist(k,l)=0; %se pone la distancia a 0
                %con respecto a cada vector de la ventana (2 minifor)
                for m=1:a
                    for n=1:b
                        if (Mask(m,n)>0) %solo comparamos respecto a los buenos
                            v2 = [Window(m,n,1) Window(m,n,2) Window(m,n,3)];
                            aux = DL2(v1,v2);
                            if (aux > MaxDL2)
                                MaxDL2 = aux;
                            end
                            dist(k,l) = dist(k,l) + aux;
                            %sumamos a la distancia acumulada
```

```
                            %pause
                        end
                    end
                end
            end
        end
    end

    Window0 = [Matrix2Vector(dist);Matrix2Vector(Window(:,:,1))
    ;Matrix2Vector(Window(:,:,2));Matrix2Vector(Window(:,:,3))];
    Window0 = sortrows(Window0');
    ADVM = Window0(1,1); %distancia acumulada del VM
    %dist
    %calculamos la posicion de minima distancia
    [imin,jmin]=find(dist==min(min(dist)));
    imin = imin(1);
    jmin = jmin(1);
    %el vector mediana segun lo anterior es ...
    VM(1)=Window(imin,jmin,1);
    VM(2)=Window(imin,jmin,2);
    VM(3)=Window(imin,jmin,3);

end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%FUNCIÓN QUE NOS DA LA MEDIDA ERROR MEDIO ABSOLUTO ENTRE LAS IMÁGENES DE  %
%                              ENTRADA.                                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function measure=MAEImages(Image1,Image2)

    [a,b,c]=size(Image1);

    N = 0;
    D = a*b*c;

    for i=1:a
        for j=1:b
            for k=1:c
                N = N + abs(double(Image1(i,j,k))-double(Image2(i,j,k)));
            end
        end
    end

    measure = N / D;

end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%FUNCIÓN QUE NOS DA LA MEDIDA DE ERROR MEDIO CUADRÁTICO ENTRE LAS         %
%                              IMÁGENES DE ENTRADA.                       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function measure=MSEImages(Image1,Image2)

    [a,b,c]=size(Image1);

    N = 0;
    D = a*b*c;

    for i=1:a
        for j=1:b
            for k=1:c
                N = N + (double(Image1(i,j,k))-double(Image2(i,j,k)))^2;
            end
        end
    end

    measure = N / D;

end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%FUNCIÓN QUE NOS DA LA MEDIDA DE PICO DE LA RELACIÓN SEÑAL RUIDO ENTRE LAS%
%                              IMÁGENES DE ENTRADA.                        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function measure=PSNRImages(Image1,Image2)

    measure = 20*log10(255/RMSEImages(Image1,Image2));

end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%     Función que nos da la fila en la que está el valor          %
%                    mínimo de una matriz                         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function[l] = minimo(Matriz)

    [x y] = min(min(Matriz));
    [m n] = size(Matriz);
    l=0;

    for j=1:n
        if Matriz(j,y) == x
            l = j;
        end
    end

end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  FUNCIÓN QUE NOS DEVUELVE LA IMAGEN SEGMENTADA EN NIVELES DE GRISES    %
%                        EN FUNCIÓN DEL UMBRAL                           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function [ImagenGrises,car] = Grises(ImagenEntrada,umbral)

[m n d] = size(ImagenEntrada);
A=ImagenEntrada;

ImagenGrises=zeros(m,n);

  for i=2:(m-1)

        for j=2:(n-1)

             B = A(i-1:i+1,j-1:j+1,1:3); %Matriz de nodos, ventanas de 3x3

             M = MatrizAdyacenciaEuclideo(B);

             %Aplicamos kruscal:

            [S,vector]=kruscalb5(M,umbral);
            [promedionodoc,car]=datos52(S,M,vector,B);
            ImagenGrises(i,j)=(car-1)*30;
        end
  end


ImagenGrises=uint8(ImagenGrises);


end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%     ALGORITMO DE KRUSKAL CON UMBRAL: ACABARÁ CUANDO UNA ARISTA      %
%                          SUPERE EL UMBRAL                           %
%                                                                     %
%    Inputs:                                                          %
%         M = matriz de adyacencia                                    %
%         umbral                                                      %
%                                                                     %
%    Outputs:                                                         %
%         S = matriz con los caminos mínimos                          %
%         vector = vector que informa de los nodos marcados y la      %
%                  componente conexa en la que están                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function[S,vector]=kruskalb5(M,umbral)

[r s] = size(M);

%vector con nodos ya conectados, cuando todo este a 1 tendremos el camino
marcados= zeros(1,r);

%creamos Matriz S inicializada a infinito, en ella iremos guardando el
%camino mínimo:

S =inf(r,s);


vector=zeros(r,2);
parada=0;
k=1;

while parada==0

    suma=0;

    [nu b]=min(min(M));
```

```
a =  minimo(M); %arista mínima

if vector(a,1) ~= 1  %a no marcado
    if vector(b,1) ~=1 %b no marcado
     c=0; %no hay ciclo
     vector(a,1)=1; % marcamos nodo
     vector(b,1)=1; % marcamos nodo
     vector(a,2)=k; %componente conexa k
     vector(b,2)=k;
     k=k+1;
    else %b marcado
      c=0; %no hay ciclo
      vector(a,1)=1; %marcamos nodo
      vector(a,2)=vector(b,2);
    end
else %a marcado
    if vector(b,1)~=1 %b no marcado
      c=0; %no hay ciclo
      vector(b,1)=1;%marcamos b
      vector(b,2)=vector(a,2);

    else
        if vector(a,2) == vector(b,2) %hay ciclo
            c=1;
        else
            c=0;

            if vector(b,2)==min(vector(a,2),vector(b,2))
                for i=1:r
                    if i ~= a && vector(i,2)== vector(a,2)

                            vector(i,2)=vector(b,2);
                    end
                end
                vector(a,2)=vector(b,2);
            elseif vector(a,2)==min(vector(a,2),vector(b,2))
```

```
                    for i=1:r
                        if i ~= b && vector(i,2)== vector(b,2)
                           vector(i,2)=vector(a,2);
                        end
                    end

                    vector(b,2)=vector(a,2);

                end
            end
        end
    end

if M(a,b)> umbral

    parada = 1; %Si la arista supera el umbral, paramos kruskal
    break;

end

if c==0 %si no hay ciclos

    S(a,b)=M(a,b);
    S(b,a) = M(a,b);
    M(a,b) = inf;
    M(b,a) = inf;
    marcados(a)=1;
    marcados(b)=1;

else %si hay ciclos, eliminamos arista de M
    M(a,b) = inf;
    M(b,a) = inf;
end

coinciden=0;
```

```
    for i=1:r
       suma = suma+marcados(i);
    end

   for j=1:r

     if vector(j,2)==vector(1,2) % para ver cuantas componentes conexas ,
        %si coindicen = numero de nodos habrá solo una componente conexa
        %y habrá que parar kruskal

         coinciden=coinciden + 1;

     end

   end


   if suma==r && coinciden == r

       parada=1;
   end



end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% FUNCIÓN QUE CALCULA EL CAMINO MÍNIMO DESDE TODOS LOS NODOS AL NODO     %
% QUE QUERAMOS UTILIZANDO EL ALGORITMO DE DIJKSTRA                       %
%                                                                        %
%    Inputs:                                                             %
%                                                                        %
%        M = Matriz de adyacencia del grafo.                             %
%        a = Nodo desde el que queremos calcular distancias mínimas.     %
%        conexas = vector con las componentes conexas del grafo.         %
%                                                                        %
%    Outputs:                                                            %
%                                                                        %
%        anterior = vector con los nodos colocados según las distancias  %
%                                                                        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function[anterior]=Dijkstrab(M,a,conexas)

%Le pasamos la matriz de adyacencia y el nodo desde el que queremos
%calcular las distancias mínimas.
%creamos un vector de tantos elementos como nodos, y metemos la distancia
%de cada nodo al nodo central a, que viene dado por la matriz de
%adyacencia. Si no estan unidos aparecerá un infinito.

    n=length(M);
    %marcados=zeros(1,n);
    for j=1:n

        marcados(j)=inf;

    end

    marcados(a)=1; %marcamos el nodo inicial

    anterior=[inf inf];

    for i=1:n
```

```
    vector(i)=inf;

end

for i=1:n

    vector2(i)=M(i,a)

end

vector=[vector;vector2];
k=2;
suma=0;

for i=1:length(conexas)
    if conexas(i,2)~= conexas(a,2)
        marcados(i)=1;
    end
end

while suma~=n

 suma=0;

 [X,Y]=min(vector(k,:));

 marcados(Y)=1; %marcamos el nodo

 for i=1:n
    if i~=Y

        vector2(i)=min(X + M(Y,i), vector(k,i));
        vector2(a)=inf;

        for j=1:n
```

```
                if marcados(j)==1

                    vector2(j)=inf;

                end
            end
        end
    end

    anterior=[anterior; X Y]; %guardamos la distancia y el nodo
    %vector2(Y)=inf;
    vector=[vector;vector2];

    k=k+1; %incrementamos k, nueva fila para vector

    for i=1:n

        suma=suma+marcados(i);

    end

end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  FUNCIÓN QUE FILTRA LA IMAGEN QUE LE PASAMOS, CONSIDERANDO DISTANCIA  %
%                          EUCLÍDEA ENTRE PÍXEL                         %
%        KRUSKAL ACABARÁ EN CUANTO UNA ARISTA SUPERE EL UMBRAL          %
%                                                                       %
%     SI QUEDA EL PÍXEL CENTRAL SÓLO, SE CAMBIARA POR EL VECTOR MEDIANA %
%           CALCULADO A PARTIR DE LA FUNCIÓN VM_DL2                     %
%                                                                       %
%    SI EL PÍXEL CENTRAL ESTA EN UNA COMPONENTE CONEXA, SE CAMBIA POR EL%
%                  PROMEDIO DE SU COMPONENTE CONEXA                     %
%                                                                       %
%     Inputs:                                                           %
%         ImagenEntrada = imagen a filtrar                              %
%         Umbral = valor de la arista a partir del cual decidimos       %
%                  cuantas zonas hay en la imagen                       %
%         lambda = peso del píxel central para la combinación lineal    %
%         p = parámetro para decidir si promediar o vector mediana      %
%                                                                       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [ImagenFiltrada, ImagenGrises,cartotal] =
             = FiltroKruscal52Dij(ImagenEntrada,umbral,lambda,p)

    A=ImagenEntrada;
    ImagenFiltrada = ImagenEntrada;
    a=5; %píxel central
    [m n d] = size(A); %m filas, n columnas
    cartotal=[];
    ImagenGrises = zeros(m,n); %Imagen en grises según las componentes conexas

    for i=2:(m-1)

        for j=2:(n-1)

             B = A(i-1:i+1,j-1:j+1,1:3); %Matriz de nodos, ventanas de 3x3
```

```
            M = MatrizAdyacenciaEuclideo(B);

            %Aplicamos kruscal:

            [S,vector] = kruscalb5(M,umbral);

            [anterior]=Dijkstrab(M,a,vector);

            [prom,card]=datos52Dij(S,M,vector,B,anterior,lambda);

            car=length(anterior);
            cartotal=[cartotal,car];

            %car tiene el cardinal de la componente conexa del pixel central

            ImagenGrises(i,j)=(car-1)*30;


            if car > p

              pixelcentral = prom;


            else

               pixelcentral = VM_DL2(B,ones(3));

            end

           ImagenFiltrada(i,j,:)= pixelcentral;
        end
    end


end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   FUNCIÓN PARA ENCONTRAR LOS MÁXIMOS EN FUNCIÓN DE LOS 3 PARÁMETROS EN EL     %
%                         FILTRO KRUSKAL-DIJKSTRA                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


    function [MPSNR,MFCSS]=  Optimizacion_parametros(A)
       lambda=0.05:0.05:0.6;
       umbral = 10:10:150;
       comp=1:5;

        MPSNR=zeros(length(umbral),length(lambda),length(comp));
        MFCSS=zeros(length(umbral),length(lambda),length(comp));

        for k=1:length(comp)
           for i=1:length(umbral)
              for j=1:length(lambda)
                  [ImagenFiltrada,PSNR,FCSS1]=KruskalDijPonderado(umbral,lambda,p,A);
                  MPSNR(i,j,p)=PSNR;
                  MFCSS(i,j,p)=FCSS1;
              end
           end
        end
    end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%FUNCIÓN QUE PASA LA MATRIZ M A UN VECTOR FILA QUE CONTIENE LAS FILAS DE  %
%               LA MATRIZ UNA A CONTINUACIÓN DE LA OTRA                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function V = Matrix2Vector(M)

    [f c] = size(M);
    V = [];

    for i=1:f

        V = [V M(i,:)];

    end

end
```

Bibliografía

[1] Gonzalez, R. C., R. E. Woods. *Digital Image Processing.* ISBN 81-203-2758-6, USA: PRENTICE HALL, 2002.

[2] Tinku Acharya, Ajoy K. Ray. *Image Procesing. Principles and Applications.* ISBN-13 978-0-471-71998-4. USA: John Wiley and Sons, 2005.

[3] Gonzalez, R. C., R. E. Woods. *Digital Image Processing Using MATLAB.* ISBN 81-7758-898-2, USA: Pearson Educational, 2006

[4] Zanuy, M. F. *Tratamiento Digital de Voz e Imagen.* ISBN 970-15-0651-0, USA: Alfaomega grupo editor, 2001.

[5] F. Martínez, A. Peris, F. Rodenas. *Tratamiento de señales digitales mediante wavelets y su uso en matlab.* ISBN 84-8454-387-0, Valencia, 2014.

[6] Sthépane Mallat. *A wavelet tour of signal processing.* ISBN 978-0-12-466606-1, USA, 1999.

[7] J.L.Gross, J.Yellen : *Graph theory and its applications*(Chapman and Hall CRC, 2006, 2nd edn.)

[8] Priyanka Kamboj, Versha Rani. *A brief study of various noise model and filtering techniques.* JGRCS, 2013, Vol.4, pp. 166-171.

[9] Ajay Kumar Boyat, Brijendra Kumar Joshi. *Noise models in digital image processing.* Signal and Image Processing : An International Journal (SIPIJ) Vol.6, No.2, April 2015.

[10] R.Maini, H.Aggarwal. *Study and Comparison of Various Image Edge Detection Techniques* . International Journal of Image Processing (IJIP), Vol.3, Iss.1.

[11] Canny, John, *A Computational Approach to Edge Detection.* Pattern Analysis and Machine Intelligence, IEEE Transactions, Vol.PAMI-8, Iss. 6.

[12] J.Fan, D.K.Y Yau, W.G Aref. *Automatic image segmentation by integrating color-edge extraction and seeded region growing.*Image Processing, IEEE Transactions, Vol.10. Iss.10.

[13] C.Jordan, S. Morillas, E.Sanabria-Codesal *Colour image smoothing through a soft-switching mechanism using a graph model*IET Image Process., 2012, Vol.6 Iss.9, pp. 1293-1298 .

[14] Schulte, S. De Witte, V.Kerre, E.E. *A fuzzy noise reduction method for colour images* IEEE Trans. Image Process., 2007, 16, pp.1425-1436.

[15] O.J. Morris, B.A., M. de J.Lee, B.Sc.(Eng.), A.C.G.I., A.G *Graph theory for image analysis: an approach based on the shortest spanning tree*, IEE Proceedings, Vol.133, Pt.F, No.2, April 1986.

[16] J.Cousty, L.Najman, F.Dias, J.Serra. *Morphological filtering on graphs.* Computer Vision and Image Understanding, 2012.

[17] C. Kenney, Y. Deng, B. S. Manjunath, and G. Hewer, *Peer Group Image Enhancement*, IEEE Transactions on image processing, Vol. 10, NO. 2, February 2001.

[18] Svetlana Grecova, Samuel Morillas, *Perceptual similarity between color images using fuzzy metrics.*

[19] E.S. Hore, B. Qiu, H.R. Wu *Noise estimation in spherical coordinates for colour image restoration.* Optical Engineering 44(4)(2005).

[20] T. Mitsa, K. Varkur, *Evaluation of contrast sensitivity functions for the formulation of quality measures incorporated in halftoning algorithms*, in: IEEE International Conference on Acoustic, Speech and Signal Processing, vol. 5,1993, pp. 301?304.

[21] Plataniotis, K.N., Venetsanopoulos, A.N. *Colour image processing and applications*, Springer-Verlag, Berlin, 2000.

[22] B. Schweizer and A. Sklar. *Statistical metric spaces*, Pacific J. Math. 10 (1960) 313?334

[23] $https : //poliformat.upv.es/portal/site/OCW_6024_2010/page/e6ae4221 - 75be - 4514 - a35c - 575a5788857b$

[24] https://www.youtube.com/playlist?list=PL6kQim6ljTJu44dsVeZifHHiuDC1MEZ7q