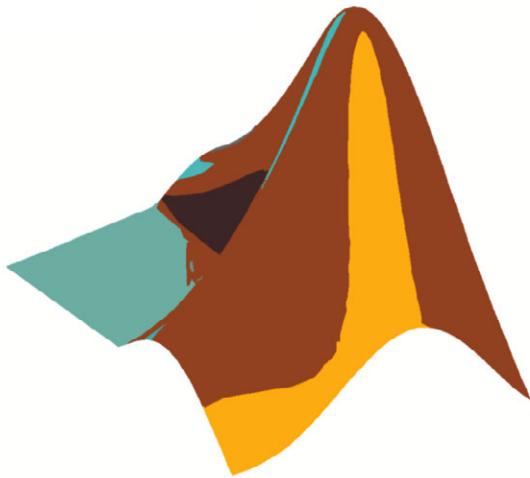




**Escuela Universitaria de Ingeniería
Técnica de Telecomunicación
Universidad Politécnica de Madrid**



Proyecto Fin de Carrera (Plan 2000)



**Descripción, comparación y ejemplos de uso
de las funciones de la toolbox de procesado
digital de imágenes de MATLAB®**

**Eduardo Laorden Fiter
Septiembre 2012**

Matlab, uno de los paquetes de software matemático más utilizados actualmente en el mundo de la docencia y de la investigación, dispone de entre sus muchas herramientas una específica para el procesado digital de imágenes. Esta toolbox de procesado digital de imágenes está formada por un conjunto de funciones adicionales que amplían la capacidad del entorno numérico de Matlab y permiten realizar un gran número de operaciones de procesado digital de imágenes directamente a través del programa principal.

Sin embargo, pese a que MATLAB cuenta con un buen apartado de ayuda tanto online como dentro del propio programa principal, la bibliografía disponible en castellano es muy limitada y en el caso particular de la toolbox de procesado digital de imágenes es prácticamente nula y altamente especializada, lo que requiere que los usuarios tengan una sólida formación en matemáticas y en procesado digital de imágenes.

Partiendo de una labor de análisis de todas las funciones y posibilidades disponibles en la herramienta del programa, el proyecto clasificará, resumirá y explicará cada una de ellas a nivel de usuario, definiendo todas las variables de entrada y salida posibles, describiendo las tareas más habituales en las que se emplea cada función, comparando resultados y proporcionando ejemplos aclaratorios que ayuden a entender su uso y aplicación. Además, se introducirá al lector en el uso general de Matlab explicando las operaciones esenciales del programa, y se aclararán los conceptos más avanzados de la toolbox para que no sea necesaria una extensa formación previa. De este modo, cualquier alumno o profesor que se quiera iniciar en el procesado digital de imágenes con Matlab dispondrá de un documento que le servirá tanto para consultar y entender el funcionamiento de cualquier función de la toolbox como para implementar las operaciones más recurrentes dentro del procesado digital de imágenes.

Matlab, one of the most used numerical computing environments in the world of research and teaching, has among its many tools a specific one for digital image processing. This digital image processing toolbox consists of a set of additional functions that extend the power of the digital environment of Matlab and allow to execute a large number of operations of digital image processing directly through the main program.

However, despite the fact that MATLAB has a good help section both online and within the main program, the available bibliography is very limited in Castilian and is negligible and highly specialized in the particular case of the image processing toolbox, being necessary a strong background in mathematics and digital image processing.

Starting from an analysis of all the available functions and possibilities in the program tool, the document will classify, summarize and explain each function at user level, defining all input and output variables possible, describing common tasks in which each feature is used, comparing results and providing illustrative examples to help understand its use and application. In addition, the reader will be introduced in the general use of Matlab explaining the essential operations within the program and clarifying the most advanced concepts of the toolbox so that an extensive prior formation will not be necessary. Thus, any student or teacher who wants to start digital image processing with Matlab will have a document that will serve to check and understand the operation of any function of the toolbox and also to implement the most recurrent operations in digital image processing.

1. Capítulo 1

Introducción y objetivos

1.1. Introducción

MATLAB es un paquete de software para el desarrollo de algoritmos, el análisis de datos, la visualización y el cálculo numérico que goza en la actualidad de un alto nivel de implantación en escuelas y centros universitarios, así como en departamentos de investigación y desarrollo de muchas compañías industriales nacionales e internacionales. En entornos universitarios, MATLAB se ha convertido en una herramienta básica tanto para los profesionales e investigadores de centros docentes como para sus alumnos.

El software cuenta con un amplio abanico de herramientas especializadas denominadas *toolboxes* que extienden significativamente la funcionalidad del programa principal. Estas herramientas cubren en la actualidad prácticamente todas las áreas principales del mundo de la ingeniería y la simulación. La *toolbox* para el procesado digital de imágenes es una de sus herramientas más famosas y utilizadas, y está formada por un conjunto de funciones que amplían las capacidades de MATLAB para el desarrollo de aplicaciones y algoritmos en el campo del procesado y análisis digital de imágenes.

La historia del procesado digital de imágenes está muy ligada al desarrollo de los computadores. Las imágenes digitales requieren tal cantidad de almacenamiento y potencia computacional que es necesario el uso de máquinas y tecnologías capaces de manejarlas. En los años 60 y 70, en paralelo al desarrollo de las aplicaciones espaciales, las técnicas de procesado digital de imágenes comenzaron también su desarrollo en diferentes campos como el de la medicina, biología, geología o astronomía. Actualmente, y gracias a los grandes avances tecnológicos de los últimos años, el procesado digital de imágenes se ha convertido en una tarea rutinaria esencial para la resolución de problemas en numerosas aplicaciones y dispositivos, y se ha acercado tanto al usuario que para obtener soluciones óptimas solo necesita entender el problema y saber aplicar las herramientas disponibles en el mercado. Superada la barrera tecnológica, la dificultad reside ahora en encontrar la documentación que permita a los usuarios poco experimentados entender el funcionamiento de los programas y técnicas involucradas en el procesado digital de imágenes.

1.2. Objetivos

La *toolbox* para el procesado digital de imágenes de MATLAB está formada por un conjunto de funciones específicas que amplían las capacidades de MATLAB para el desarrollo de aplicaciones y algoritmos en el campo del procesado y análisis digital de imágenes. Pese a que existen manuales de usuario del programa en español creados por la comunidad educativa (no oficiales), no existe documentación específica para la *toolbox* de procesado digital de imágenes y la poca información existente se reduce al manual oficial del programa en inglés y a bibliografía altamente especializada que requiere que los usuarios tengan ya una sólida formación en matemáticas y en procesado digital de imágenes.

El objetivo de este trabajo es el de definir, analizar y comparar a nivel de usuario las funciones incluidas en la *toolbox* para el procesado digital de imágenes de MATLAB, proporcionando ejemplos aclaratorios que ayuden a entender su uso y aplicación, y que faciliten su elección y ejecución para resolver problemas determinados. Con ello, sin entrar en conceptos avanzados de programación o procesado digital de imágenes, tanto el profesional de la docencia como el alumno que se inicie en el campo del procesado digital de imágenes con MATLAB, dispondrá de la ayuda y base teórica fundamental para el entendimiento de la herramienta y el desarrollo futuro de programas y aplicaciones más complejas.

2. Capítulo 2

MATLAB y el procesado digital de imágenes

2.1. ¿Qué es MATLAB?

MATLAB (abreviatura de *MATRIX LABoratory*, "laboratorio de matrices") es un software matemático que ofrece un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio (lenguaje M). Está disponible para las plataformas Unix, Windows y Mac OS X.

Entre sus prestaciones básicas se encuentran la manipulación de matrices, la representación de datos y funciones, la implementación de algoritmos, la creación de interfaces de usuario (GUI) y la comunicación con programas en otros lenguajes y con otros dispositivos hardware.^[1]

El paquete de software de MATLAB dispone de dos herramientas adicionales que expanden sus prestaciones: Simulink (plataforma de simulación multidominio) y GUIDE (editor de interfaces de usuario - GUI). Además, se pueden ampliar las capacidades de MATLAB con las cajas de herramientas (*toolboxes*) que cubren actualmente la mayoría de áreas principales del mundo de la ingeniería y la simulación, como por ejemplo la *toolbox* de procesado digital de imágenes (IPT de aquí en adelante), la *toolbox* de procesado de señal, la *toolbox* de comunicaciones... y así hasta más de 150 herramientas.

MATLAB proporciona un entorno de trabajo interactivo cuyo elemento básico de trabajo son las matrices, lo que permite la resolución numérica de problemas en un tiempo mucho menor que si se utilizaran lenguajes de programación tradicionales como pueden ser los lenguajes Fortran, Basic o C, con la ventaja de que su lenguaje propio de programación es similar al de los lenguajes tradicionales. Al trabajar con matrices se pueden describir infinidad de variables de una forma altamente flexible y matemáticamente eficiente. Por ejemplo, una imagen se puede escribir como una matriz de píxeles, un sonido como una matriz de fluctuaciones, y en general se puede describir con una matriz cualquier relación lineal entre las componentes de un modelo matemático.

MATLAB empezó a utilizarse por investigadores y profesionales de Ingeniería de Control, expandiéndose rápidamente a diferentes disciplinas. Actualmente es un programa muy utilizado en educación, particularmente en la enseñanza del álgebra lineal y el análisis numérico, y es muy popular entre los científicos trabajando en procesado digital de imágenes. Los usuarios de MATLAB proceden de varios campos de la ingeniería, ciencia y economía, y pese a ser un paquete de software profesional y de precio elevado, ha conseguido instaurarse rápidamente en las aulas y despachos de muchos centros educativos de todo el mundo con una política efectiva de licencias de evaluación de bajo coste.

2.2. Introducción al Procesado Digital de Imágenes (PDI)

2.2.1. Definición

El procesado digital de imágenes (PDI) es el procesado, entendiendo éste como el almacenamiento, transmisión y representación de información, de imágenes digitales por medio de una computadora digital.

El término imagen se refiere a una función bidimensional de intensidad de luz $f(x, y)$ donde x e y denotan las coordenadas espaciales, y el valor de f en cualquier punto (x, y) es proporcional a la intensidad de la imagen en ese punto. Una imagen digital puede escribirse como una matriz cuyos índices de fila y columna identifican un punto en la imagen y cuyo valor coincide con el nivel de intensidad de luz en ese punto. Cada elemento del array se corresponde con un elemento en la imagen y se le denomina pixel.

El interés en el procesado digital de imágenes se basa esencialmente en dos aspectos: la mejora de la información contenida en una imagen para la interpretación humana y el tratamiento de los datos de una escena para la favorecer la percepción autónoma por parte de una máquina.

Debido al amplio rango de tipos de imágenes empleadas en el PDI, no existe un límite claro respecto dónde se encuentra la línea divisoria entre el PDI y otras áreas afines, como el análisis de imágenes o la visión por computador, entre otras. El análisis de imágenes se refiere al proceso por el cual se extrae información cuantitativa de la imagen y en donde el resultado del análisis es siempre una tabla de datos, una gráfica o cualquier representación de los datos numéricos. El procesado de imágenes, sin embargo, siempre produce otra imagen como resultado de la operación, por lo que por lo general se pretende

mejorar la calidad de una imagen para poder apreciar mejor determinados detalles. La visión por computador o visión artificial es un subcampo de la inteligencia artificial cuyo propósito es programar un computador para que "entienda" una escena o las características de una imagen, emulando la visión humana.^[2]

Atendiendo a los tipos de procesos implicados en estas disciplinas, se suele hacer una clasificación de tres niveles: nivel bajo (procesado), nivel medio (análisis) y nivel alto (interpretación). Los procesos de nivel bajo incluyen la reducción de ruido, la mejora del contraste, y en general, la mejora de características de la imagen, en donde todas las entradas/salidas son imágenes. Los procesos de nivel medio analizan los niveles bajos e incluyen la segmentación (regiones, objetos), descripción de objetos, clasificación, etc. La entrada es una imagen y la salida son atributos de los objetos (bordes, contornos, identidades de objetos individuales). Por último, los procesos de nivel alto están generalmente orientados al proceso de interpretación de los elementos obtenidos en los niveles inferiores, entrando en juego el entendimiento y la toma de decisiones en función del contenido observado. Con esta clasificación, se le llama procesado digital de imágenes a los procesos cuyas entradas y salidas son imágenes (procesos de bajo nivel) y, además, a aquellos procesos que extraen atributos de imágenes, incluyendo el reconocimiento de objetos individuales (procesos de nivel medio).

2.2.2. Orígenes

Los antecedentes históricos del procesado digital de imágenes se remontan a la impresión de periódicos en 1921. En aquella época la codificación y transmisión de datos se realizaba por cable submarino entre las ciudades de Londres y Nueva York, en donde se reconstruía e imprimía. En 1922 se mejoró el proceso al emplear una técnica basada en la reproducción fotográfica a través de cintas perforadas en las terminales telegráficas receptoras que permitía obtener 5 niveles de gris. Hacia 1929 la técnica se mejoró de nuevo hasta obtener 15 niveles de gris en la reproducción de una fotografía. A pesar de estos avances, las imágenes que se produjeron con esas técnicas no se consideran los inicios del PDI debido a que su creación no involucró el uso de la computadora. Hubo que esperar hasta los años 60, con la llegada de los grandes ordenadores y del programa espacial estadounidense para ver las primeras técnicas de procesado digital de imágenes por computador. Concretamente hasta el año 1964 en el Laboratorio de Propulsión de la

NASA, cuando se procesaron las imágenes de la Luna enviadas por el satélite Ranger 7 para corregir distorsiones propias de la cámara.

A partir de los años 60 el PDI no ha parado de beneficiarse de los continuos avances tecnológicos, entre los que se encuentran principalmente la invención del transistor en los Laboratorios Bell en 1948, el desarrollo de los lenguajes de programación de alto nivel, la invención del Circuito Integrado por Texas Instrument en 1958, el desarrollo de los sistemas operativos, la introducción del ordenador personal en 1981 (IBM), y la continua miniaturización de componentes y desarrollo de sistemas de almacenamiento.

2.2.3. Áreas de aplicación

Además de su aplicación en los programas espaciales, las técnicas de procesado digital de imagen comenzaron a utilizarse en una gran variedad de ámbitos y problemas que compartían la necesidad de mejora de las imágenes para su interpretación y análisis.

Actualmente existe una inmensa gama de áreas donde el PDI se utiliza de manera habitual. Un criterio de clasificación habitual para diferenciar cada una de las áreas implicadas es el de la fuente de las imágenes. La principal fuente de energía de las imágenes es el espectro electromagnético (EM), especialmente la banda de rayos X y la del espectro visible. Otras fuentes son la acústica, la ultrasónica, y la electrónica.

Espectro electromagnético.

Longitud de onda (λ) en metros.

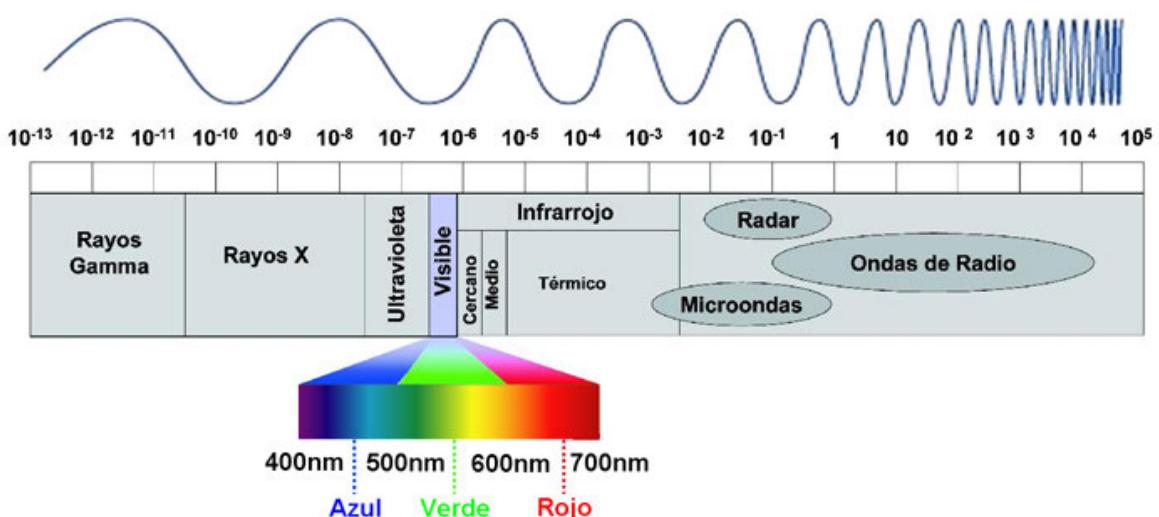


Figura i.1 - Espectro electromagnético

Empezando por los rayos Gamma, existen aplicaciones que procesan esas fuentes de imagen en medicina nuclear, como en la Tomografía por Emisión de Positrones (PET) donde se le inyecta al paciente un isótopo radioactivo que emite rayos gamma que posteriormente son capturados, y en las observaciones astronómicas para detectar la radiación natural de rayos gamma de los astros. La siguiente banda, la de los rayos X, tiene sus principales aplicaciones en medicina (radiografía, angiografía, Tomografía Axial Computarizada (TAC), etc.), pero también en astronomía e industria, como en el control de calidad de placas de circuito impreso o de alimentos.

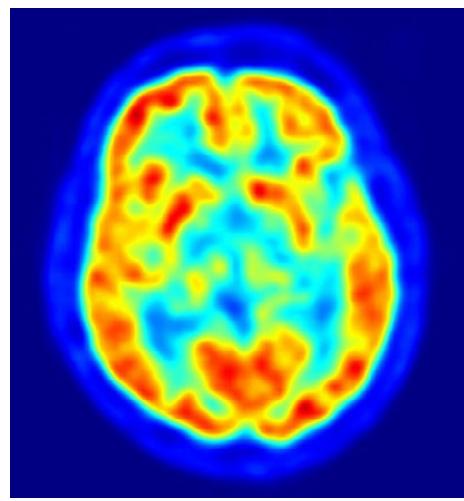


Figura i.2 - Escáner PET del cerebro humano^[3]

También es posible obtener imágenes empleando la banda ultravioleta del espectro electromagnético, teniendo como principales aplicaciones la litografía, la inspección industrial, la microscopía, los láseres, las imágenes biológicas y la astronomía.

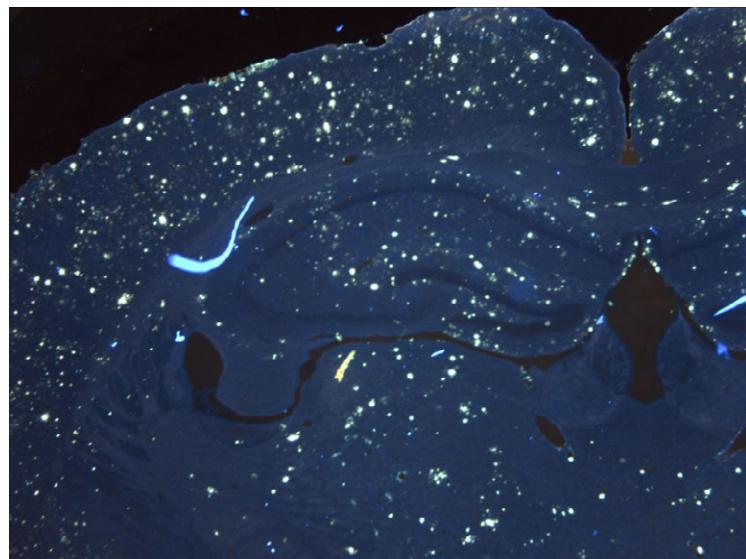


Figura i.3 - Teñido del cerebro de un ratón visto en un microscopio de fluorescencia con luz ultravioleta.

Respecto a las imágenes captadas en la banda visible e infrarroja, estas son por mucho las aplicaciones más numerosas. La banda infrarroja se utiliza habitualmente en conjunto con la banda de espectro visible, por ejemplo en microscopía, astronomía, detección remota (con varias bandas que detectan diferentes detalles del terreno cada una), controles de calidad en la industria y en las aplicaciones de identificación de la policía.



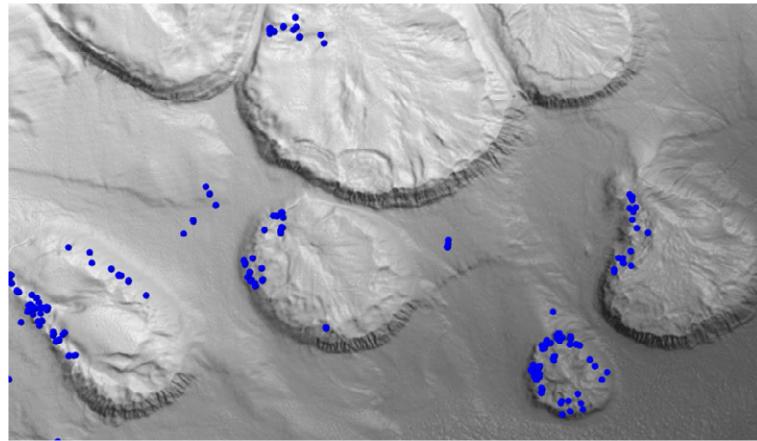
Figura i.4 - Detección de vegetación con imagen infrarroja

Las imágenes de la banda de microondas tienen al radar como aplicación principal ya que este puede obtener datos de virtualmente cualquier región y en cualquier momento, sin importar condiciones de clima o de luz ambiental (puede penetrar nubes, vegetación, hielo, arena) utilizando su propia “iluminación” (pulsos de microondas) para obtener una imagen a partir de la energía reflejada. Las imágenes de la banda de radio tienen su mayor aplicación en astronomía y medicina. En medicina las ondas de radio se utilizan principalmente en Imágenes de Resonancia Magnética (MRI).



Figura i.5 - Montañas en el Tíbet vistas por un radar topográfico (NASA)

Otras formas de obtener imágenes, en este caso imágenes acústicas, es a través del sonido. Se aplica en exploración geológica (petróleo y minerales), en la industria y en medicina. El ejemplo más notorio es el ultrasonido, con el que se puede realizar un seguimiento al embarazo, o captar imágenes de partes del cuerpo como la tiroides o las capas musculares.



*Figura i.6 - Detección de emisiones de gases en el fondo del Golfo de México con sonar.
Créditos: Universidad de New Hampshire Center.*

En las imágenes electrónicas, como las de la microscopía electrónica, la microscopía de escaneo de electrones (SEM) y la microscopía de transmisión de electrones (TEM), se utilizan rayos de electrones enfocados en vez de rayos de luz como los microscopios ópticos. A través de esta técnica es posible captar un filamento de tungsteno o detectar un circuito integrado dañado.

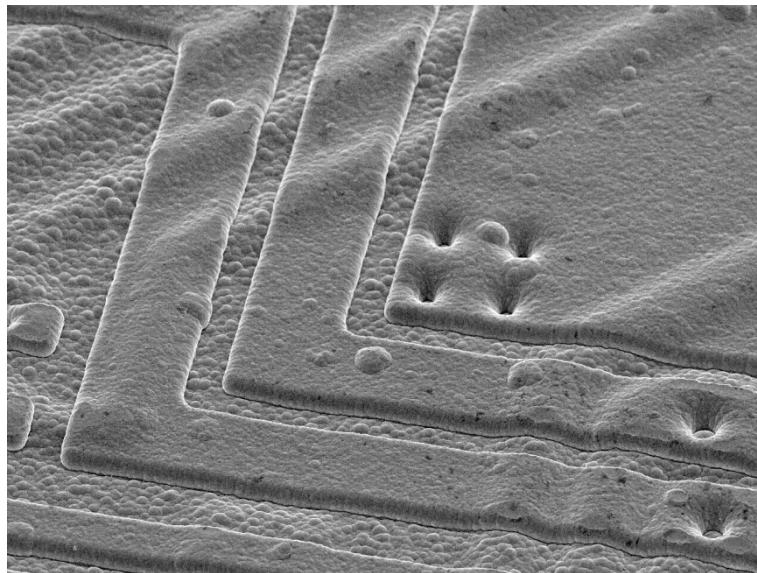


Figura i.7 - Circuito integrado ampliado 2000 veces con SEM

2.2.4. Clases principales de procesado digital de imágenes

Conviene clasificar los diferentes procesos involucrados en el procesado digital de imágenes para tener una visión general más estructurada. Podemos englobar la mayor parte de tareas en tres categorías, cada una de ellas con diferentes algoritmos involucrados:

- **Mejora o realce de la imagen:** procesado de la imagen para que el resultado sea más apropiado para una aplicación en particular. Tareas habituales: mejora de la nitidez o aclarado de imágenes desenfocadas, eliminación del ruido, mejora del contraste, mejora del brillo, detección de bordes, etc.
- **Restauración de la imagen:** Se puede considerar como revertir el daño ocasionado a la imagen por una causa conocida. Tareas habituales: eliminar el desenfoque por movimiento, eliminar distorsiones ópticas, eliminar interferencia periódica, etc.
- **Segmentación de la imagen:** Subdivide la imagen en partes o aísla ciertos objetos de una imagen. Tareas habituales: búsqueda y selección de formas determinadas en la imagen, máscaras de la imagen, etc.

Estas clases suelen aparecer en algún momento del procesado de la imagen aunque no siempre deben hacerlo y dependerá del objetivo y del problema específico en cada caso.

2.2.5. Pasos fundamentales en el procesado digital de imágenes

La complejidad de la tarea determinará el número de procesos necesarios para resolver el problema, pero los pasos fundamentales y más habituales en el procesado digital de imágenes son los siguientes:

1. **Adquisición de la imagen:** los sensores digitalizan las señales. Generalmente incluye pre-procesado de la imagen (p.ej. escalar la imagen).
2. **Mejora de la imagen:** procesos de manipulación de la imagen para lograr un resultado más adecuado que el original para una aplicación específica (obtener detalles que no se veían, o simplemente destacar ciertas características de interés).
3. **Restauración de la imagen:** también mejora la apariencia de la imagen pero a diferencia de la mejora de la imagen que es subjetiva, la restauración es objetiva en el sentido en que las técnicas de restauración tienden a ser modelos probabilísticos o matemáticos de degradación de la imagen (¿cómo era la imagen antes de estropearse?)

4. **Procesado del color:** determinado color puede dar o resaltar más información.
5. **Ondículas (wavelets):** utilizadas fundamentalmente para representar imágenes en varios grados de resolución. Se utiliza principalmente en compresión.
6. **Compresión:** reduce el almacenamiento requerido para guardar una imagen, o el ancho de banda para transmitirla.
7. **Operaciones morfológicas:** herramientas para extraer componentes de la imagen útiles para la representación y descripción de formas.
8. **Segmentación:** divide una imagen en sus partes constituyentes. Los objetos se extraen o aíslan del resto de la imagen para su posterior análisis. Es una de las tareas más difíciles del procesado digital de imágenes.
9. **Representación y descripción:** casi siempre recibe una imagen segmentada que consta solamente de fronteras o de regiones. Se toman decisiones tales como si la forma obtenida debe ser tratada como un frontera o una región, y se extraen atributos que resultan en información cuantitativa de interés o que son básicos para diferenciar clases de objetos.
10. **Reconocimiento:** el proceso que asigna una etiqueta (p. ej. “vehículo”) a un objeto basándose en sus descriptores o bien le da un significado a un grupo de objetos ya reconocidos. ^[4]

2.3. Introducción al Procesado Digital de Imágenes con MATLAB

2.3.1. Introducción

Para poder utilizar cualquiera de las diferentes *toolboxes* especializadas de las que dispone MATLAB es necesario tener un conocimiento mínimo del programa y de sus funciones más básicas, comunes a todas las herramientas. El objetivo de este trabajo no es el de ahondar en la utilización del programa principal, ya que existe suficiente bibliografía como para que el usuario pueda adquirir los fundamentos del programa sin demasiada dificultad. Además, muchas de las funciones y capacidades del programa principal no son necesarias para poder iniciarse en el procesado digital de imágenes con MATLAB, así que nos estaríamos extralimitando en nuestro contenido. Por tanto, a continuación introduciremos solamente las funciones y operaciones generales de MATLAB que resulten fundamentales para iniciarse en la lectura de este trabajo y en el procesado digital de imágenes con MATLAB.

2.3.2. El entorno de MATLAB

El entorno de MATLAB incluye tres elementos principales: ventanas, variables y ficheros.

- **Ventanas:** Son de diversos tipos. Las ventanas que forman el núcleo (kernel) del programa se organizan en el escritorio (*desktop*), pero en una sesión típica se abren y cierran gran número de ventanas secundarias correspondientes a figuras, editores de ficheros o de variables. Existen además ventanas específicas correspondientes a la ayuda y a las demostraciones (*helps* y *demos*). En el procesado digital de imágenes aparecerán sobre todo ventanas con las imágenes que van siendo tratadas a lo largo de los programas.
- **Variables:** Son objetos temporales (al cerrar MATLAB se borran) que durante la sesión en curso se almacenan en el espacio de trabajo (*workspace*).
- **Ficheros:** Son objetos permanentes (al cerrar MATLAB no se borran). Destacan los ficheros tipo M (.m), específicos de MATLAB, que guardan funciones o simplemente código ejecutable directamente en el programa.

2.3.3. El escritorio de MATLAB

Al abrir MATLAB desde el sistema operativo pertinente aparece un escritorio (*desktop*) como el mostrado en la figura siguiente (versión 7.13).

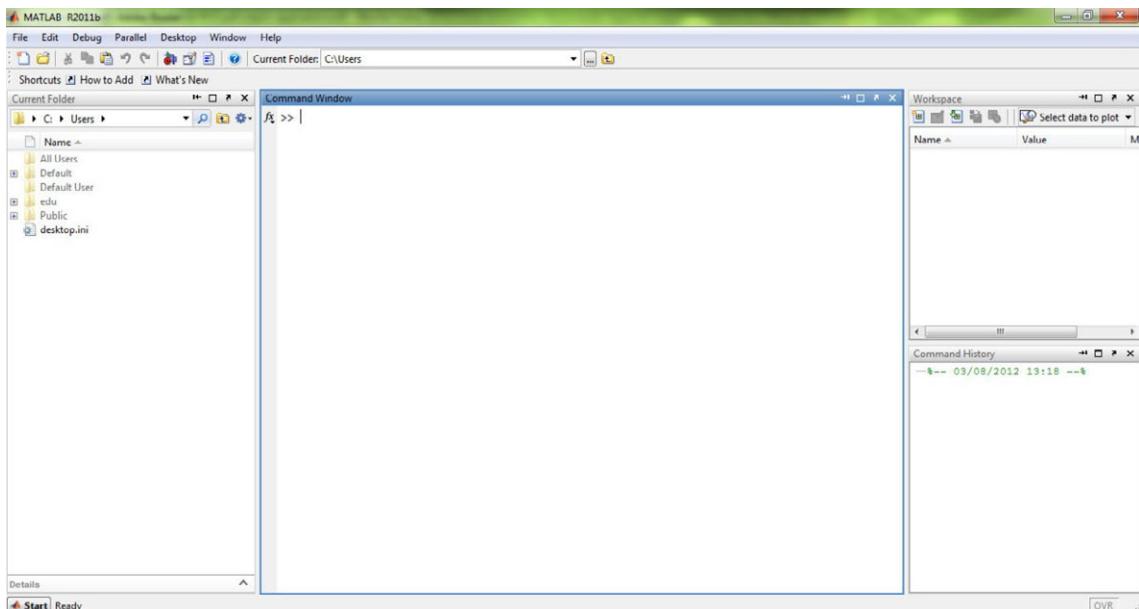


Figura i.8 - Escritorio de MATLAB

En él se pueden diferenciar las siguientes zonas de interés, de izquierda a derecha y de arriba a abajo:

- **Directorio actual (Current Folder)**: es el directorio donde se ejecutarán los comandos, pudiendo leer y escribir datos de/en ese directorio. En la ventana se listan los ficheros y carpetas del directorio actual.
- **Ventana de comandos (Command Window)**: es la ventana principal. Desde ella se escriben los comandos de MATLAB. Las instrucciones se escriben después del *prompt*, cuyo símbolo es `>>` y se ejecutan con la tecla Enter. Se puede acceder a las instrucciones ejecutadas con anterioridad pulsando las teclas de flecha izquierda y derecha para no tener que escribirlas de nuevo.
- **Espacio de trabajo (Workspace)**: muestra las variables creadas durante la ejecución del programa, así como su tipo y su valor. Haciendo doble clic en ellas se puede abrir el visualizador/editor de variables.
- **Historial de comandos (Command History)**: almacena las instrucciones introducidas en cada una de las sesiones anteriores de MATLAB, indicando fecha y hora de la sesión.

Algunos comandos útiles para el manejo del escritorio de MATLAB y sus ventanas son los siguientes:

`clc`

Borrar la ventana de comandos (todos los comandos escritos).

`close [all] [all hidden]`

Cerrar todas las ventanas abiertas de figuras, aplicaciones, etc.

`exit, quit`

Salir de MATLAB. Pulsar CTRL+C para detener algún cálculo, gráfico o impresión sin salir del programa.

`diary [on/off]`

Grabar una sesión en un fichero de texto.

`hold [on/off]`

Retener la gráfica de la ventana actual y que los siguientes comandos se apliquen sobre ella.

`ver`

Mostrar la versión de MATLAB y de las *toolboxes* instaladas.

`help nombre_comando`

Obtener ayuda sobre cualquier comando.

2.3.4. Variables y ficheros

Durante una sesión, las variables creadas por los comandos se guardan en el espacio de trabajo, donde pueden ser modificadas y/o utilizadas en otros comandos. Para ver el contenido de una variable basta con teclear su nombre en la ventana de comandos o ir a la ventana *Workspace* para hacer doble clic en el nombre de la variable y abrir el editor de variables.

Es posible guardar las variables del *workspace* en un fichero llamado `matlab.mat` para que no se pierdan al cerrar MATLAB. Se hace con la función `save`, para poder cargarlas posteriormente en otra sesión con la función `load`. El comando `diary` permite guardar el texto de la sesión actual en el fichero de texto especificado.

Así como las variables son la información temporal de MATLAB, los ficheros constituyen la información permanente que no se borra al cerrar la sesión. Hay 2 tipos de ficheros principales: ficheros de datos con extensión `*.mat`, que contienen datos de distintos formatos y se crean y cargan ejecutando las instrucciones `save` y `load` (o a través de la ventana *Workspace*) y ficheros M con extensión `*.m`, que contienen funciones con parámetros de entrada y salida o simplemente comandos ejecutables. Normalmente cada función de MATLAB es en realidad un fichero tipo M en cuyo interior está escrita la función.

MATLAB también puede importar datos de otros tipos de ficheros (`*.txt`, `*.xls`,...). Si el fichero está dentro del directorio actual de trabajo, bastará con hacer clic sobre él y seleccionar la opción de Importación o bien importar los datos directamente desde la ventana de comandos con funciones específicas para ello, como `importdata`, `fopen`, `fread`, etc.

Los ficheros creados por el usuario se guardan por defecto en el directorio de trabajo, que es el directorio sobre el que se ejecuta cada sesión de MATLAB. Si se tiene algún fichero en otro lugar y se quiere hacer uso de su contenido, se le debe indicar a MATLAB que busque en dicho directorio mediante el comando `addpath` directorio.

Algunos comandos útiles para el manejo de variables y ficheros son:

`clear [var][all]`: borrado de variables.

`what`

Ver qué ficheros se encuentran en el directorio actual.

`lookfor`

Buscar funciones o comandos relacionados con la palabra especificada.

`size`

Leer el tamaño de una matriz en filas y columnas.

`ndims`

Leer las dimensiones de una matriz (un vector tiene 2 al ser una matriz de 1 x 1).

`whos`

Mostrar información adicional de una o todas las variables actuales.

`length`

Leer la longitud de un vector, equivalente a `max(size(X))`.

2.3.5. Comandos y matrices

La forma directa de interactuar con MATLAB es a través de la línea de comandos. Los comandos se escriben después del símbolo `>>` y se ejecutan presionando la tecla Enter. Si una instrucción no cabe en una línea, se escriben 3 puntos seguidos y se puede continuar en la línea siguiente. Los comentarios se hacen precedidos del carácter `%` y no se ejecutan.

Si no se asigna una variable a la ejecución de un comando, el resultado del comando se asigna automáticamente a la variable `ans` (de *answer*). Para que MATLAB no muestre el resultado de una instrucción esta debe terminar con `:` (punto y coma).

MATLAB no copia el contenido de una variable en otra salvo que la nueva variable vaya a ser modificada. Por ejemplo, al escribir `B=A` no se estará copiando el contenido de `A` en `B` hasta que el contenido de `B` cambie posteriormente en el programa.

MATLAB trabaja esencialmente con matrices numéricas rectangulares y la manera más fácil de escribir matrices pequeñas es enumerando sus elementos de tal manera que:

- Los elementos estén separados por espacios en blanco o comas.
- Los elementos estén escritos entre corchetes, [].
- Se defina el final de cada fila con ; (punto y coma).

por ejemplo:

```
A = [ 1 2 3; 4 5 6; 7 8 9 ]
```

daría la matriz de salida:

```
A =  
1 2 3  
4 5 6  
7 8 9
```

Si la matriz a introducir es muy grande se puede utilizar el siguiente formato, escribiendo cada fila en una nueva línea:

```
A = [ 1 2 3  
4 5 6  
7 8 9 ]
```

Las matrices pueden estar formadas por un solo elemento (escalar), por una fila o una columna (vector, una dimensión) o por una serie de filas y columnas (matriz, array de dos dimensiones). Al hablar de arrays en términos generales se hace referencia a cualquier conjunto de números del mismo tipo, definido entre corchetes, cuyos elementos pueden ser indexados a través de enteros, lo que incluye a las matrices y los vectores. Se habla de matriz dispersa cuando es una matriz de gran tamaño en la que la mayor parte de sus elementos son cero. Por ejemplo:

```
A = 1  
define a A como un escalar de valor 1.
```

```
A = [ 1 2 3 ]  
define a A como un vector o array fila de tres elementos: A(1)=1, A(2)=2 y A(3)=3.
```

```
A=[ 1, 2, 3; 4, 5, 6 ]  
define a A como una matriz o array de dos dimensiones.
```

Los arrays de celdas son arrays multidimensionales cuyos elementos son copias de otros arrays y que se suelen utilizar al manejar números y caracteres a la vez. Por ejemplo, el array de celdas `c = {'gauss', [1 0; 0 1], 3}` contiene 3 elementos: una cadena de caracteres, una matriz de 2-por-2 y un escalar, a los que se puede acceder individualmente mediante `c{1}`, `c{2}` o `c{3}` respectivamente. Importante es que estos arrays trabajan con copias de las matrices que reciben como entrada, no con punteros a dichas matrices.

Las estructuras son otro tipo de variable similar a los arrays de celdas en el sentido de que permiten agrupar una colección de datos distintos en una única variable, pero en este caso se accede a los campos mediante el nombre del campo seguido del nombre de la

variable. Por ejemplo en la estructura: `a = struct('campo1', 0, 'campo2', 'val')` se accede a cada uno de los campos mediante `a.campo1` y `a.campo2`.

Los dos puntos `(:)` es uno de los operadores más utilizados en MATLAB, permitiendo crear vectores, subarrays y especificar iteraciones. Básicamente significa “en todo el rango”.

`j:k`
es igual que `[j, j+1, ..., k]`

`j:i:k`
es igual que `[j, j+i, j+2i, ..., k]`

`A(:, j)`
es la columna número `j` de `A` (de todas sus filas)

`A(:, j:k)`
es `A(:, j), A(:, j+1), ..., A(:, k)`

Para acceder a los diferentes elementos de las matrices podemos hacerlo de la siguiente manera:

`A(1,1)`
si queremos el elemento `(1,1)` de la matriz.

`A(1:2,3)`
si queremos las dos primeras filas de la tercera columna escribiremos

`A(3,:)`
si queremos todos los elementos (todas las columnas) de la tercera fila escribiremos

`B=A(1:2,2:3)`
`B=A([1,2],[2 3])`
si queremos crear la submatriz `B` formada por las filas 1 a 2 y las columnas 2 a 3 de la matriz `A`.

Respecto a las operaciones matemáticas básicas con matrices y vectores, los signos que se utilizan son: `+` para sumar, `-` para restar, `*` para multiplicar, `'` para trasponer y `^` seguido del número al que se pretende elevar la matriz para potenciar. Poner un punto antes de un operador indica que la operación debe realizarse componente a componente. Por ejemplo, `A*B` es el producto matricial entre las matrices `A` y `B`. Si se teclea `C=A.*B` (siendo `A` y `B` matrices de igual dimensión), el resultado es una matriz `C` cuyas componentes c_{ij} corresponden a $c_{ij}=a_{ij}\times b_{ij}$.

Algunos comandos útiles para el manejo y creación de matrices son:

`zeros(N), zeros(M,N)`

Matriz de ceros de N-por-N o M-por-N de clase double.

`y=zeros(size(A))`

Matriz de ceros del tamaño de A.

`ones(N), ones(M,N)`

Matriz de unos de N-por-N o M-por-N de clase double.

`true(N), true(M,N)`

Matriz de unos de N-por-N o M-por-N de clase logical.

`false(N), false(M,N)`

Matriz de ceros de N-por-N o M-por-N de clase logical.

`magic(M)`

“Cuadrado mágico” de M-por-M. Es un array cuadrado en donde la suma a lo largo de una fila, columna o diagonal principal, es la misma. Es un array muy utilizado para probar resultados.

`rand(N), rand(M, N)`

Matriz de N-por-N o M-por-N con sus valores distribuidos uniformemente con números aleatorios en el intervalo [0, 1].

`randn(N), randn(M,N)`

Matriz de N-por-N o M-por-N con sus valores distribuidos uniformemente con números de media 1 y varianza 0.

2.3.6. Representación de imágenes

Como ya hemos comentado anteriormente, podemos definir una imagen como una función bidimensional de intensidad de luz $f(x, y)$, donde x e y denotan las coordenadas espaciales y el valor de f en cualquier punto (x, y) es proporcional a la intensidad de la imagen en ese punto. En caso de que la imagen sea en escala de grises se habla de nivel de gris. Las imágenes en color son una combinación de imágenes individuales, como en el espacio de trabajo típico RGB, donde se combinan tres componentes: roja, verde y azul. Por tanto las operaciones sobre imágenes en escala de grises se pueden aplicar también a imágenes en color procesando cada componente por separado.

Al digitalizar una imagen se digitalizan las coordenadas mediante muestreo y los valores de amplitud mediante cuantificación. De esta forma, se obtiene una matriz de números reales de tamaño M-por-N.

El sistema de coordenadas utilizado en la IPT usa la notación (r, c) para indicar las filas y las columnas, con el origen de coordenadas situado en (r, c) = (1, 1). La mayor parte

de la bibliografía de procesado digital de imágenes utiliza sin embargo la notación (x, y) con el centro en $(f, y) = (0, 0)$.

2.3.7. Entrada y salida de imágenes

2.3.7.1. Lectura de imágenes

Las imágenes se leen desde la línea de comandos de MATLAB mediante la función `imread`, incorporando así al espacio de trabajo su variable/array correspondiente:

```
I = imread('nombre_archivo');
```

Si la imagen se encuentra en el directorio actual de trabajo no hará falta indicar la ruta al archivo. Sin embargo, para imágenes fuera del directorio actual de trabajo, necesitaremos escribir la ruta completa al archivo.

Mediante la función `size` podemos ver las dimensiones de la imagen. Con la siguiente sintaxis recogemos el número de filas y columnas en las variables `F` y `C`:

```
[F, C] = size(I)
```

Y mediante la función `whos` podemos ver varias de sus características, como sus dimensiones, bytes ocupados y clase de archivo:

```
whos I
```

Para tener acceso a todos los campos de información de una imagen se debe recoger la estructura de información devuelta por la función `iminfo` de la siguiente manera:

```
E = iminfo(I);
```

y posteriormente acceder a los diferentes campos de la estructura como se ha comentado con anterioridad:

```
tamaño = E.FileSize;
```

2.3.7.2. Visualización de imágenes

Para visualizar una imagen se debe utilizar la función `imshow`. Al ejecutar la función se abrirá la imagen especificada en una nueva ventana, sin producir una salida de datos en la línea de comandos. Si se ejecuta de nuevo la función sin haber cerrado la

ventana o sin especificar una nueva figura utilizando el comando `figure` antes de llamar a la función (`figure, imshow(I)`) la nueva imagen reemplazará la anterior en la ventana abierta. Las sintaxis más habituales son las siguientes:

`imshow(I, G)`

donde `I` es la imagen a mostrar y `G` es el número de niveles de intensidad utilizados para mostrarla (256 si se omite).

`figure, imshow(I)`

donde `I` es la imagen a mostrar, que se abrirá en una nueva ventana.

`imshow(I, [low high])`

donde `I` es la imagen a mostrar y donde se mostrarán como negro todos los valores menores o iguales que el valor `low` y como blanco todos los valores mayores o iguales que el valor `high`, con los valores intermedios mostrados por intensidad según el número de niveles por defecto.

`imshow(I, [])`

donde `I` es la imagen a mostrar y donde el mínimo valor del array se usará como `low` y el máximo como `high` para mostrar el mínimo valor como negro y el máximo como blanco. Es útil para mostrar imágenes con bajo rango dinámico o con valores de intensidad positivos y negativos.

2.3.7.3. Escritura de imágenes

Las imágenes se guardan o escriben en el disco mediante la función `imwrite`. Si no se especifica otro directorio en el nombre del archivo la función guarda la imagen en el directorio actual de trabajo. Las sintaxis más habituales son:

```
imwrite(f, 'nombre_archivo')
imwrite(f, 'nombre_archivo', 'tipo_imagen')
imwrite(f, 'nombre_archivo.extensión')
```

Para determinados tipos de archivo existen opciones adicionales, como en el caso de los archivos tipo JPG, donde se puede especificar la calidad del archivo final mediante la variable `q`, que debe valer entre 0 y 100:

```
imwrite(f, 'nombre_archivo.jpg', 'quality', q)
```

Para guardar gráficos generados con MATLAB debemos hacerlo a través del menú contextual de la ventana, seleccionando “*File-Save As...*” o bien mediante la función `print` desde la línea de comandos con la siguiente sintaxis:

```
print -fnumfig -dformato -rresolucion nombre_archivo
```

Por ejemplo, para guardar la figura número 1 en el archivo 'prueba' en formato TIFF y con resolución de 300 puntos por pulgada deberemos escribir:

```
print -f1 -dtiff -r300 prueba
```

2.3.8. Tipos de datos y de imágenes

2.3.8.1. Tipos de datos ^[5]

Existen diferentes tipos o clases de datos en MATLAB, divididos en tres categorías: numéricos, de caracteres y lógicos. La siguiente tabla define cada uno de ellos:

Nombre	Descripción
double	Números de precisión doble en el rango aproximado de -10^{308} a 10^{308} . Si es una imagen con valores escalados (lo habitual) el rango será [0, 1] (8bytes por elemento)
uint8	Enteros sin signo de 8-bits en el rango [0, 255] (1 byte por elemento)
uint16	Enteros sin signo de 16-bits en el rango [0, 65535] (2 bytes por elemento)
uint32	Enteros sin signo de 32-bits en el rango [0, 4294967295] (4 bytes por elemento)
int8	Enteros con signo de 8-bits en el rango [-128, 127] (1 byte por elemento)
int16	Enteros con signo de 16-bits en el rango [-32768, 32767] (2 bytes por elemento)
int32	Enteros con signo de 32-bits en el rango [-2147483648, 2147483647] (4 bytes por elemento)
single	Números de precisión simple en el rango aproximado de -10^{38} a 10^{38} (4 bytes por elemento)
char	Caracteres (2 bytes por elemento)
logical	Valores 0 o 1 (1 byte por elemento)

Para la conversión de datos se utilizan las siguientes funciones de la IPT (más detalladas en la página dedicada a cada función):

Función	Convierte a clase:	Clases de entrada
im2uint8	uint8	logical, uint8, uint16, double
im2uint16	uint16	logical, uint8, uint16, double

<code>mat2gray</code>	double escalada (rango [0, 1])	double
<code>im2double</code>	double	logical, uint8, uint16, double
<code>im2bw</code>	logical	uint8, uint16, double

2.3.8.2. Tipos de imágenes

La *toolbox* soporta 4 tipos de imágenes: de intensidad, binarias, RGB e indexadas. La mayoría de operaciones en el procesado digital de imágenes monocromáticas se llevan a cabo utilizando imágenes binarias o de intensidad. A continuación se describe cada tipo de imagen por separado:

- **Imágenes de intensidad:** son matrices de datos cuyos valores se han escalado para representar una escala determinada de intensidad. Por ejemplo, cuando sus elementos son de clase `uint8`, tendrán valores numéricos entre 0 y 255.
- **Imágenes binarias:** son arrays de ceros y unos de clase `logical` (lógica o binaria). Un array de ceros y unos cuyos valores no son de clase lógica (`uint8` por ejemplo), no se considera una imagen binaria. Para convertir un array numérico a un array binario se debe utilizar la función `logical`. Para comprobar si un array es binario se utiliza la función `islogical`.
- **Imágenes RGB (Red Green Blue):** son arrays de píxeles de color de tamaño `M`-por-`N`-por-3, donde cada pixel está formado por una tripleta de valores correspondientes a las componentes roja, verde y azul de la imagen en la posición determinada. El tipo de dato de las componentes determina el rango de valores. Por ejemplo, si la imagen RGB es de clase `double`, su rango irá de 0 a 1, o si es de clase `uint8`, tendrá valores numéricos entre 0 y 255.

El número de bits utilizados para representar los valores de los píxeles en cada componente de la imagen determinan la profundidad de color de la imagen RGB. Por ejemplo, si cada componente de la imagen es de 8 bits, la imagen RGB será de 24 bits de profundidad.

Si se quiere crear una imagen RGB a partir de sus componentes por separado se debe utilizar la función `cat` para juntarlas. Para extraer cada una de las componentes de una imagen RGB se puede utilizar la siguiente sintaxis:

```
R = RGB(:, :, :, 1);  
G = RGB(:, :, :, 2);  
B = RGB(:, :, :, 3);
```

- **Imágenes indexadas:** tienen dos componentes: una matriz de datos formada por números enteros y una matriz `map` con el mapa o paleta de color. La matriz del mapa de color es un array de `m`-por-3 de clase `double` que contiene valores en el rango [0, 1] y donde la longitud de `m` es igual al número de colores que define. Cada valor de la matriz del mapa de color contiene las componentes roja, verde y azul que definen un único color. El color de cada pixel en una imagen indexada se determina utilizando el valor de la matriz de enteros como puntero (índice) al array del mapa de color. De esta forma, el valor de la matriz de enteros determina el número de fila en la matriz del mapa de color (un color determinado). Por ejemplo, si la matriz de enteros es de clase `double`, todas las componentes con valor menor o igual que 1 apuntan a la primera fila del mapa de color, las componentes con valor 2 apuntan a la segunda fila y así sucesivamente. Para mostrar una imagen indexada se debe utilizar la sintaxis:

```
imshow (x, map)
```

o alternativamente:

```
image(x)  
colormap(map)
```

Existen diferentes mapas de color predefinidos que pueden utilizarse para mostrar la imagen.

Para pasar de un tipo de imagen a otro la IPT proporciona varias funciones que permiten la conversión. Estas son las más habituales (más detalladas en la página de cada función):

Función	Imagen de salida	Imagen de entrada
<code>dither</code>	imagen indexada (mediante trámado)	imagen RGB
<code>grayslice</code>	imagen indexada (mediante umbralización multinivel)	imagen de intensidades en escala de grises
<code>gray2ind</code>	imagen indexada	imagen de intensidades en escala de grises
<code>ind2gray</code>	imagen de intensidades en	imagen indexada

	escala de grises	
rgb2ind	imagen indexada	imagen RGB
ind2rgb	imagen RGB	imagen indexada
rgb2gray	imagen en escala de grises	imagen RGB

2.3.9. Indexado de arrays

2.3.9.1. Indexado de vectores

Como ya definimos al comienzo de este capítulo, un vector v es un array de una sola dimensión, ya sea horizontal (1-por-N) o vertical (N-por-1). Para acceder a cada uno de los elementos del vector v se utiliza indexado en una dimensión: $v(1)$ es el primer elemento del vector, $v(2)$ es el segundo elemento, y así sucesivamente.

Recordamos que para acceder a bloques de elementos se utiliza el operador : (dos puntos) de tal forma que si escribimos $v(1:3)$ estaremos accediendo a los primeros tres elementos del vector. Para indicar el final del vector podemos hacerlo con la palabra end de la siguiente manera: $v(1:end)$, lo que produciría la salida del vector como fila. Para escribir el vector como columna deberemos escribir $v(:)$.

Se puede acceder a posiciones del vector no sucesivas con la sintaxis: $v(1:2:end)$ que significa coger el elemento 1, contar de dos en dos para coger elementos y finalizar cuando se llegue al final del vector. Los saltos pueden ser negativos (cuenta hacia atrás).

Un vector también se puede utilizar como índice en otro vector. Por ejemplo, podemos coger los elementos primero, cuarto y quinto del vector v utilizando el vector índice [1 4 5] con la siguiente sintaxis: $v([1 4 5])$.

Mediante la función `linspace(a, b, n)` se puede generar un vector fila de n elementos linealmente espaciados y que incluya a los números a y b .

2.3.9.2. Indexado de matrices

Para indexar matrices ahora necesitamos dos índices al ser elementos en dos dimensiones: uno para la fila y otro para la columna. Por ejemplo, para acceder a la segunda fila y tercera columna del vector A se debe de escribir: $A(2, 3)$.

El uso del operador : (dos puntos) es similar que en el caso de vectores pero ahora se seleccionan bloques enteros de filas o columnas especificadas. Por ejemplo, siendo A una matriz de 3-por-3, seleccionaremos únicamente la tercera columna mediante el comando `A(:, 3)` o bien `A(1:3, 3)`. Para escribir la matriz como una columna deberemos escribir `A(:)`, que es una forma adecuada para utilizar con funciones como `sum` que suma todas las columnas de la matriz, por lo que el comando `sum(A(:))` calculará la suma de todos los elementos del array. Para escribir la matriz como fila deberemos escribir `A(1:end)`.

El uso de la variable `end` también es similar que en el indexado de vectores, salvo que ahora se indicará el final de la fila o columna. Por ejemplo `A(end, end)` mostrará el último elemento del array.

La notación `A([a b], [c d])` selecciona los elementos de A que están en la fila a columna c, fila a columna d, fila b columna c y fila b columna d. Una forma habitual de indexar matrices con matrices es utilizar matrices binarias que hagan de índice como muestra el siguiente ejemplo:

```
A = [1 2 3; 4 5 6; 7 8 9]
D = logical([1 0 0; 0 0 1; 0 0 0])
A(D)
```

que daría una salida de :

```
ans =
    1
    6
```

MATLAB guarda las matrices del modo en el que las muestra el comando `A(:)`, o sea, en una sola columna. De hecho, se puede acceder a los elementos de la matriz mediante indexado lineal, utilizando la posición del elemento en esa única columna. Por ejemplo, `A(3)` accede al tercer valor de esa columna de datos.

2.3.10.Operadores

Los operadores de MATLAB se agrupan en tres categorías principales:

- Operadores aritméticos que realizan operaciones numéricas.
- Operadores relacionales que comparan operandos cuantitativamente.
- Operadores lógicos.

2.3.10.1. Operadores aritméticos

MATLAB cuenta con dos tipos de operaciones aritméticas: operaciones matriciales y operaciones de arrays. La diferencia es que las operaciones de arrays se realizan elemento a elemento y se especifican con el operador . (punto). Por ejemplo, $A*B$ indica una multiplicación de matrices tradicional mientras que $A.*B$ indica la multiplicación de arrays (de sus elementos). Para los operadores de suma y resta no se utiliza el operador punto.

La tabla siguiente muestra los operadores aritméticos de MATLAB, donde A y B son matrices o arrays (pudiendo ser imágenes) y a y b son escalares. Se necesitan entradas de clase `double`:

Operador	Operación	Función
<code>+</code>	Suma de arrays o matrices.	<code>plus(A, B)</code>
<code>-</code>	Resta de arrays o matrices.	<code>minus(A, B)</code>
<code>.*</code>	Multiplicación de arrays.	<code>times(A, B)</code>
<code>*</code>	Multiplicación de matrices.	<code>mtimes(A, B)</code>
<code>./</code>	División de cada entrada de A por su correspondiente en B (división de array derecha).	<code>rdivide(A, B)</code>
<code>.\</code>	División de cada entrada de B por su correspondiente en A (división de array izquierda).	<code>ldivide(A, B)</code>
<code>/</code>	División de matrices derecha (A/B es similar a $A*inv(B)$).	<code>mrdivide(A, B)</code>
<code>\</code>	División de matrices izquierda ($A\backslash B$ es similar a $inv(A)*B$).	<code>mldivide(A, B)</code>
<code>.^</code>	Potenciación de arrays.	<code>power(A, B)</code>
<code>^</code>	Potenciación de matrices.	<code>mpower(A, B)</code>
<code>.'</code>	Traspuesta de vectores y matrices.	<code>transpose(A)</code>
<code>'</code>	Adjunta o traspuesta conjugada de vectores y matrices.	<code>ctranspose(A)</code>

Además, la IPT tiene una serie de funciones específicas para operar con imágenes que aunque realizan cálculos que se pueden llevar a cabo con los operadores aritméticos tradicionales, su entrada de datos no está limitada a la clase `double`.

Función	Descripción
<code>imadd</code>	Suma dos imágenes o le añade una constante a una imagen.
<code>imsubtract</code>	Resta dos imágenes o le resta una constante a una imagen.
<code>immultiply</code>	Multiplica dos imágenes elemento a elemento o multiplica una constante sobre una imagen.

<code>imdivide</code>	Divide dos imágenes elemento a elemento o divide una imagen entre una constante.
<code>imabsdiff</code>	Calcula la diferencia absoluta entre dos imágenes.
<code>imcomplement</code>	Complementa una imagen.
<code>imlincomb</code>	Combina linealmente dos o más imágenes.

2.3.10.2. Operadores relacionales

Los operadores relacionales de los que dispone MATLAB son los habituales y comparan elemento a elemento los elementos correspondientes de cada array, que deben ser de iguales dimensiones salvo que uno de ellos sea un escalar. Los operadores son los siguientes:

Operador	Operación lógica
<code><</code>	Menor que.
<code><=</code>	Menor o igual que.
<code>></code>	Mayor que.
<code>>=</code>	Mayor o igual que.
<code>==</code>	Igual que.
<code>~=</code>	Distinto que.

Este tipo de comandos son más habituales en conjunción con sentencias de programación (`if`, `while`, `for`, etc.) pero también pueden utilizarse directamente con arrays, lo que devolverá un array lógico indicando los valores de los arrays que cumplen con las condiciones lógicas. Ejemplo:

```
A = [1 2 3; 4 5 6; 7 8 9];
B = [0 2 4; 3 5 6; 3 4 9];
A == B
```

dará la siguiente salida (marcando dónde A es igual a B):

```
ans =
0     1     0
0     1     1
0     0     1
```

2.3.10.3. Operadores y funciones lógicas

Los operadores lógicos de MATLAB pueden operar tanto con datos lógicos como con datos numéricos. El valor verdadero o 1 es cuando hay un valor lógico de 1 o cuando hay un valor numérico distinto de cero.

Operador	Operación lógica
&	AND
	OR
NOT	NOT

Hay 3 funciones lógicas importantes en MATLAB, que son:

Función	Descripción
xor	OR exclusiva. Devuelve 1 si ambos operandos son diferentes lógicamente y 0 en caso contrario.
all	Devuelve 1 si todos los elementos de un vector son distintos de cero y 0 en caso contrario. Esta función opera con columnas en el caso de matrices.
any	Devuelve 1 si alguno de los elementos de un vector es distinto de cero y 0 en caso contrario. Esta función opera con columnas en el caso de matrices.

Ejemplo:

```
A = [ 1 2 3; 4 5 6];
B = [ 0 -1 1; 0 0 2];

all(B)
```

daría una salida de :

```
ans =
    0      0      1
```

2.3.11. Programación de archivos

Una de las principales características de la IPT es su total acceso al entorno de programación de MATLAB, tanto desde la línea de comandos como desde los archivos-M propios de MATLAB con código y funciones ejecutables.

El lenguaje de programación de MATLAB es similar al empleado en otros lenguajes de alto nivel como puede ser C o Java, y las expresiones fundamentales para el manejo del flujo de comandos son las mismas (*if*, *else*, *for*, *switch*, *while*, *break*, *continue*, *return*, etc.).

Además existe una serie de funciones lógicas que son útiles para la programación condicionada y que se listan a continuación:^[5]

Función	Descripción
iscell(C)	Verdadero si C es un array de celdas.

<code>iscellstr(s)</code>	Verdadero si <code>s</code> es un array de celdas de cadenas de texto.
<code>ischar(s)</code>	Verdadero si <code>s</code> es un carácter.
<code>isempty(A)</code>	Verdadero si <code>A</code> es un array vacío [].
<code>isequal(A,B)</code>	Verdadero si <code>A</code> y <code>B</code> tienen idénticos elementos y dimensiones.
<code>isfield(S, 'nombre')</code>	Verdadero si ' <code>nombre</code> ' es un campo de la estructura <code>S</code> .
<code>isfinite(A)</code>	Verdadero en las localizaciones del array <code>A</code> que son finitas.
<code>isinf(A)</code>	Verdadero en las localizaciones del array <code>A</code> que son infinitas.
<code>isletter(A)</code>	Verdadero en las localizaciones de <code>A</code> que son letras del alfabeto.
<code>islogical(A)</code>	Verdadero si <code>A</code> es un array lógico.
<code>ismember(A,B)</code>	Verdadero en las posiciones donde los elementos de <code>A</code> están también en <code>B</code> .
<code>isnan(A)</code>	Verdadero en las posiciones de <code>A</code> que son números indeterminados (NaNs, <i>Not-a-Number</i>) como p.ej. 0/0.
<code>isnumeric(A)</code>	Verdadero si <code>A</code> es un array numérico.
<code>isprime(A)</code>	Verdadero en las posiciones <code>s</code> de <code>A</code> que son números primos.
<code>isreal(A)</code>	Verdadero si los elementos de <code>A</code> no tienen parte imaginaria.
<code>isspace(A)</code>	Verdadero en las localizaciones donde los elementos de <code>A</code> son espacios en blanco.
<code>issparse(A)</code>	Verdadero si <code>A</code> es una matriz dispersa.
<code>isstruct(S)</code>	Verdadero si <code>S</code> es una estructura.

3. Capítulo 3

La *toolbox* de procesado digital de imágenes de MATLAB

1.1. Introducción

La Image Processing Toolbox™ (IPT) le proporciona a MATLAB un conjunto completo de algoritmos y herramientas gráficas para el procesado, análisis y visualización de imágenes y para el desarrollo de aplicaciones y de nuevos algoritmos en el campo del procesado y análisis de imágenes digitales. Además MATLAB, el entorno matemático sobre el que se sustenta, es ideal para este procesado digital de imágenes ya que las imágenes digitales son matrices al fin y al cabo.

La IPT soporta imágenes generadas por un amplio gama de dispositivos, incluidas cámaras digitales, sensores satelitales y aéreos, dispositivos de generación de imágenes médicas, microscopios, telescopios y otros instrumentos científicos. La herramienta soporta formatos de datos e imágenes estándar, como JPEG, JPEG-2000, TIFF, PNG, HDF, HDF-EOS, FITS, Microsoft® Excel®, ASCII, y archivos binarios, pero también admite diversos formatos de imagen especializados, como los formatos de imágenes multibanda BIP y BIL utilizados por los satélites LANDSAT, los archivos médicos DICOM y sus metadatos asociados, o los formatos Analyze 7.5 e Interfile. La *toolbox* también puede leer imágenes geoespaciales en archivos NITF e imágenes de alto rango dinámico en archivos HDR.

Entre las funciones principales de la IPT destacan:^[6]

- Mejora y filtrado de imágenes y enfoque de imágenes borrosas.
- Análisis de imágenes, incluyendo segmentación, morfología, extracción de funciones y medición.
- Transformaciones geométricas y métodos de registro de imágenes basados en intensidad.
- Transformaciones de imágenes, incluyendo FFT, DCT, Radon y proyección de haz de rayos en abanico.

- Flujos de trabajo para procesar, visualizar y navegar por imágenes arbitrariamente grandes.
- Herramientas interactivas, incluyendo selecciones de ROI, histogramas y mediciones de distancias.
- Importación y exportación de archivos DICOM.

1.2. Funciones ^[7]

La IPT de MATLAB cuenta con 283 funciones propias en su última versión 8.0 (R2012a) del año 2012. En este capítulo y según lo acordado con el tutor del PFC, se describirá únicamente la primera mitad de las funciones para no exceder el tiempo y espacio que se le debe destinar a la realización del PFC, quedando la otra mitad pendiente para otro futuro PFC que complementemente esta primera mitad del trabajo.

A continuación se detallará sucesivamente cada función por separado, empezando alfabéticamente por la primera y utilizando una nueva página para cada una de ellas, facilitando así su lectura y comprensión.

En cada función se seguirá una estructura similar: el nombre de la función, su título descriptivo, una introducción en caso de que sea necesario aclarar brevemente algunos conceptos avanzados, sus sintaxis permitidas, una descripción completa de la función y de sus diferentes sintaxis y variables, y unos ejemplos finales aclaratorios. Al ser un programa y código completamente en inglés, se ha decidido dejar en inglés el nombre de las variables implicadas en la definición de las funciones, aunque su explicación posterior se haga en castellano. Todas las palabras que formen parte del código de un programa o de la sintaxis de una función tendrán un formato de fuente distinto al resto para facilitar su diferenciación.

Al final de la página de cada función se detallan las funciones de la IPT que están relacionadas con esa función. Además, en el Capítulo 5 - Anexo I se encuentran organizadas las funciones por aplicación para que sea más fácil encontrar una función a partir de los requisitos del usuario. De esta forma, junto al índice de funciones ordenadas alfabéticamente localizado al comienzo de este PFC, se hace sencilla la búsqueda de cualquier función en el documento, ya sea por aplicación o por nombre.

adaphisteq

Ecualización adaptativa del histograma con contraste limitado (Contrast-limited adaptive histogram equalization, CLAHE)

Introducción

El histograma es un gráfico que relaciona los niveles de intensidad de gris de una imagen y el número de píxeles que hay para cada nivel de intensidad. El histograma, además de caracterizar la imagen, se puede utilizar también para la segmentación de imágenes.

Cada imagen tiene su propio histograma pero como norma general se considera que una imagen tiene un buen contraste si su histograma se extiende ocupando casi todo el rango de tonos. Para aumentar el contraste, se puede expandir el histograma o bien realizar una ecualización del mismo (expansión buscando distribución uniforme):

- Expansión del histograma (o aumento del rango dinámico): consiste en hacer que el histograma abarque la totalidad de niveles de intensidad disponibles, expandiendo los niveles originales de manera que el valor de intensidad más bajo sea cero y el más alto sea el valor máximo.
- Ecualización o linealización del histograma: busca distribuir los niveles de gris (ó color en el caso RGB) de una forma uniforme, expandiendo los valores de intensidad más frecuentes en la imagen y por tanto mejorando el contraste de la imagen. La principal ventaja es que es un método completamente “automático”. [4]

La función `adaphisteq` mejora el contraste de la imagen `I` utilizando una ecualización adaptativa del histograma con contraste limitado (CLAHE). Es útil para potenciar los detalles de una imagen y mejorar el contraste tanto general como local. Es la función más sofisticada y automática de la *toolbox* para mejorar el contraste de una imagen. Otras alternativas más simples son las funciones `histeq`, que ecualiza el histograma actuando sobre la imagen completa e `imadjust` que permite expandir el histograma ajustando los valores de intensidad de la imagen a un rango determinado.

Sintaxis

```
J = adapthisteq(I)
J = adapthisteq(I,param1,val1,param2,val2...)
```

Descripción

CLAHE actúa en pequeñas regiones de la imagen llamadas ‘baldosas’ (*tiles*) en vez de hacerlo en toda la imagen. Se mejora el contraste de cada baldosa para que el histograma de la región concuerde de manera aproximada con el histograma definido en el parámetro ‘Distribution’. Después se combinan las baldosas vecinas usando interpolación bilineal para eliminar las fronteras inducidas artificialmente. Esto es especialmente útil cuando se necesita aplicar una transformación diferente en distintas zonas de la imagen, aumentando el contraste de forma diferente en cada zona, cosa que no se puede hacer con las otras funciones de la toolbox destinadas a la mejora del contraste (`histeq` e `imadjust`).

El contraste, especialmente en las zonas homogéneas, se puede limitar para evitar amplificar cualquier ruido que pudiera presentar la imagen en esas zonas. Con ello evitamos sobreexponer zonas que ya están bien contrastadas.

I es una imagen en escala de grises y puede ser del tipo `uint8`, `uint16`, `int16`, `single`, o `double`. La imagen de salida *J* tendrá el mismo tipo que *I*. Si en vez de una imagen en escala de grises se tiene una imagen en color, se debe convertir la imagen en color al espacio de color adecuado para procesarlo, como por ejemplo al espacio de color $L^*a^*b^*$, donde se puede actuar únicamente sobre la componente de luminosidad ‘ L^* ’, afectando solo a la intensidad de los píxeles y preservando los colores originales.

Opcionalmente se pueden especificar varios parámetros de entrada, cuyos valores por defecto funcionan muy bien para una primera aproximación o prueba. Los nombres de los parámetros se pueden abbreviar y no son sensibles a las mayúsculas:

Parámetro	Valor
'NumTiles'	Vector de dos números enteros positivos que especifican el número de baldosas rectangulares con fila y columna, [M N]. M y N deben valer al menos 2. El número total de baldosas es igual a M*N. <code>adapthisteq</code> dividirá la imagen en ese número de regiones o baldosas y calculará la transformación de contraste para cada una de ellas basándose en su histograma. El número óptimo depende del tipo de imagen de entrada y se debe determinar por experimentación. (Por defecto: [8 8], 64 baldosas)
'ClipLimit'	Escalar real en el rango [0, 1] que define un límite de mejora en el

	<p>contraste. A mayor número mayor contraste. Es el parámetro que más afecta al resultado final.</p> <p>'ClipLimit' es un factor de contraste que previene una sobresaturación de la imagen, especialmente en zonas homogéneas, con similar nivel de gris. Estas zonas se caracterizan por tener un pico en el histograma debido a demasiados píxeles con mismo nivel de gris. Sin esta limitación, la ecualización adaptativa del histograma podría producir una imagen peor que la original.</p> <p>(Por defecto: 0.01)</p>
'NBins'	<p>Escalar entero positivo que especifica el número de intervalos (<i>bins</i>) del histograma usado para crear la transformación de contraste mejorado. A mayor número, mejor rango dinámico, a cambio de menor velocidad de procesado.</p> <p>(Por defecto: 256)</p>
'Range'	<p>Cadena que especifica el rango de datos de la imagen de salida.</p> <p>'original' — Rango limitado al de la imagen original, $[\min(I(:)) \max(I(:))]$.</p> <p>'full' — Rango completo del tipo de imagen usado en <i>I</i>. Por ejemplo, si se usa un tipo <i>uint8</i>, el rango será [0, 255].</p> <p>(Por defecto: 'full')</p>
'Distribution'	<p>Cadena que especifica la forma o distribución deseada del histograma que usará la función para realizar la transformación. La distribución depende del tipo de imagen de entrada. Por ejemplo, las imágenes submarinas quedan más naturales utilizando una distribución Rayleigh.</p> <p>'uniform' — Histograma Uniforme o plano.</p> <p>'rayleigh' — Histograma en forma de campana.</p> <p>'exponential' — Histograma exponencial o curvado.</p> <p>(Por defecto: 'uniform')</p>
'Alpha'	<p>Escalar real positivo que especifica un parámetro de distribución. Solo se usa cuando se ha especificado el parámetro 'Distribution' como 'rayleigh' o 'exponential'.</p> <p>(Por defecto: 0.4)</p>

Ejemplos de uso

Ejemplo 1: Aplicar una ecualización adaptativa del histograma con contraste limitado (CLAHE) a una imagen en escala de grises cambiando alguno de sus parámetros.

```
I = imread('M131806467LC.tif');
imshow(I)
```

% Utilizamos la función con diferentes parámetros y comparamos.

```
A1 = adapthisteq(I);
A2 = adapthisteq(I,'clipLimit',0.05);
A3 = adapthisteq(I,'NumTiles',[30 30],'clipLimit',0.01);
figure, imshow(A1)
figure, imshow(A2)
figure, imshow(A3)
```



Figura f.1 - Imagen original (*I*) sin procesar

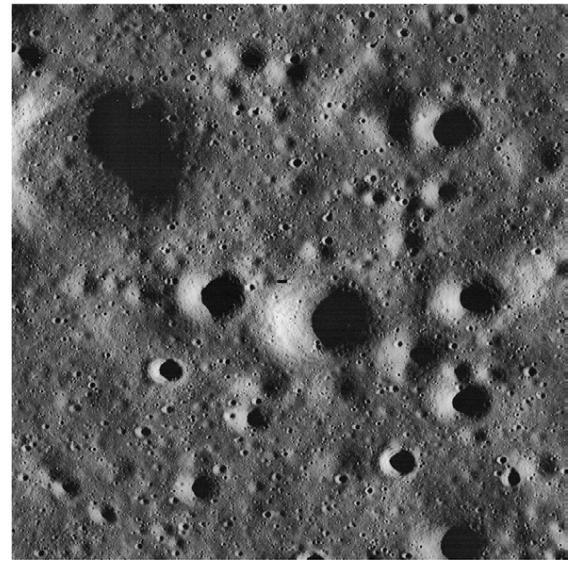


Figura f.2 - Imagen procesada (*A1*) con *adapthisteq* con los parámetros por defecto (*NumTiles*=8*8 y *clipLimit*=0.01)

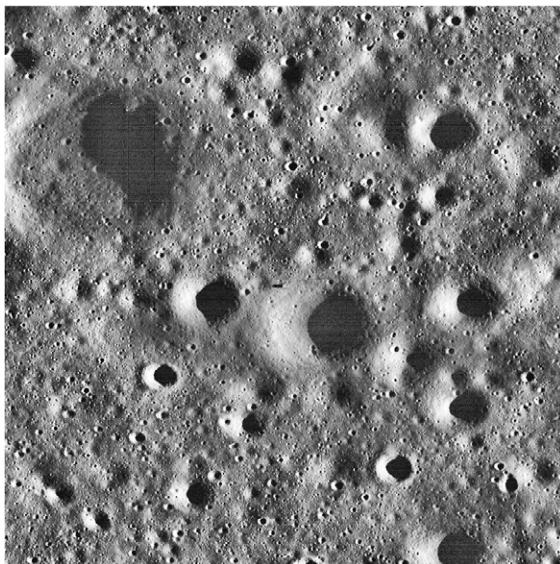


Figura f.3 - Imagen procesada (*A2*) con *adapthisteq* con los parámetros por defecto y *clipLimit*=0.05

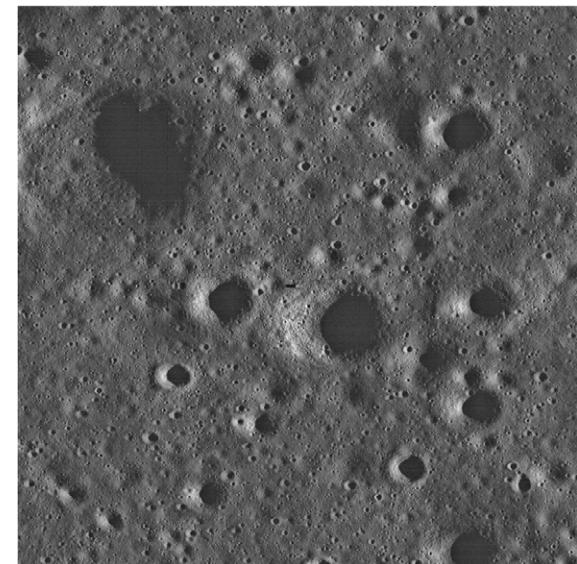


Figura f.4 - Imagen procesada (*A3*) con *adapthisteq* con los parámetros por defecto y *NumTiles*=30*30

Ejemplo 2: Aplicar una ecualización adaptativa del histograma con contraste limitado (CLAHE) comparando el resultado con el de las otras funciones para la mejora del contraste (histeq e imadjust).

```
I = imread('M131806467LC.tif');
figure, imhist(I);
A1 = imadjust(I);
figure, imhist(A1);
A2 = histeq(I);
figure, imhist(A2);
A3 = adapthisteq(I);
figure, imhist(A3);

imshow(I)
figure, imshow(A1)
figure, imshow(A2)
figure, imshow(A3)
```

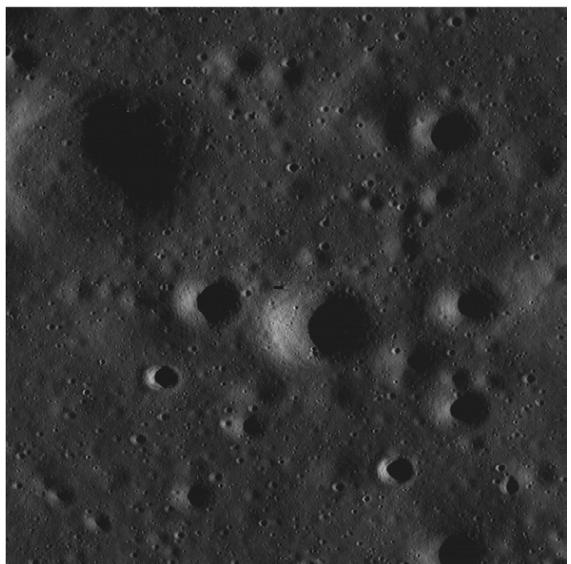


Figura f.1 - Imagen original (*I*) sin procesar

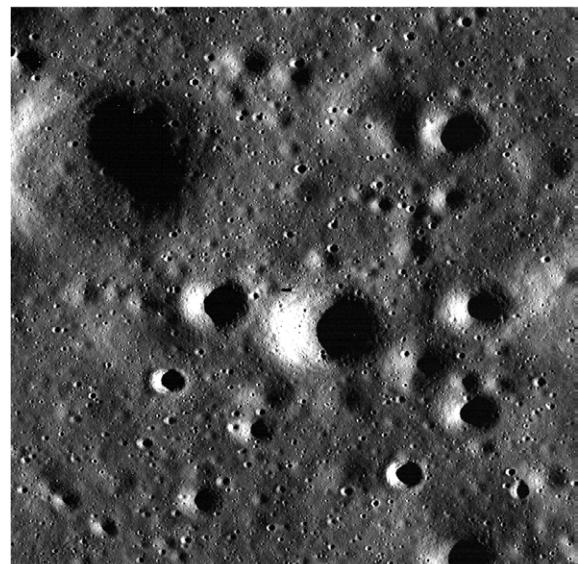


Figura f.5 - Imagen procesada (*A1*) con *imadjust*

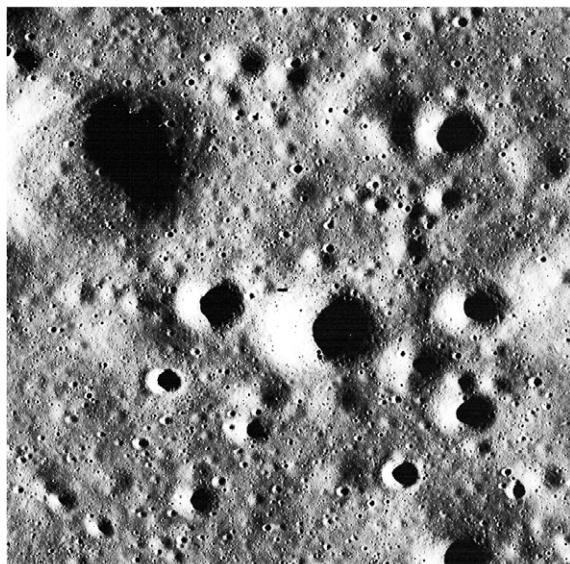


Figura f.6 - Imagen procesada (*A2*) con *histeq*

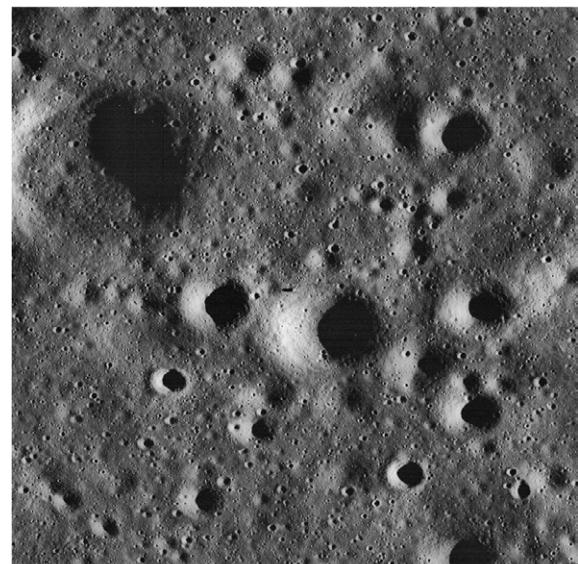


Figura f.2 - Imagen procesada (*A3*) con *adapthisteq*

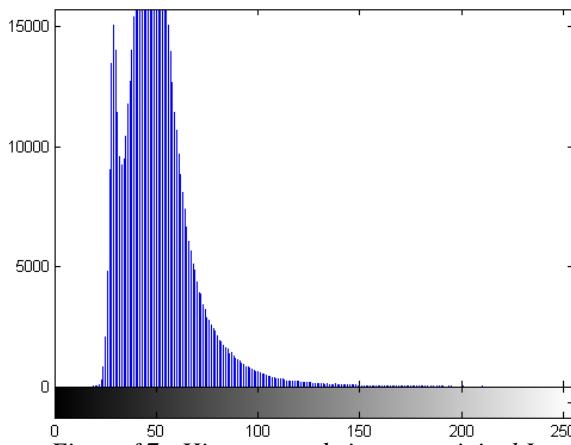


Figura f.7 - Histograma de imagen original I

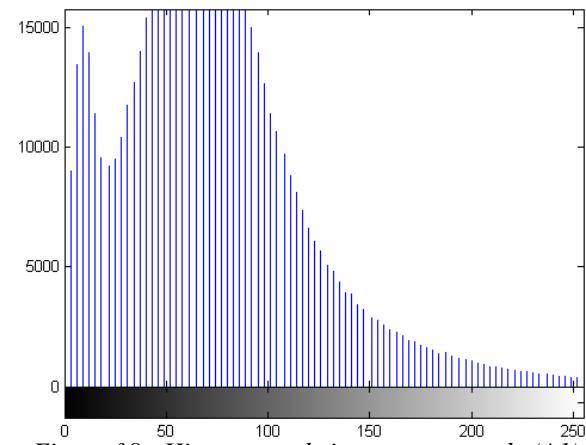


Figura f.8 - Histograma de imagen procesada (A1)
con imadjust

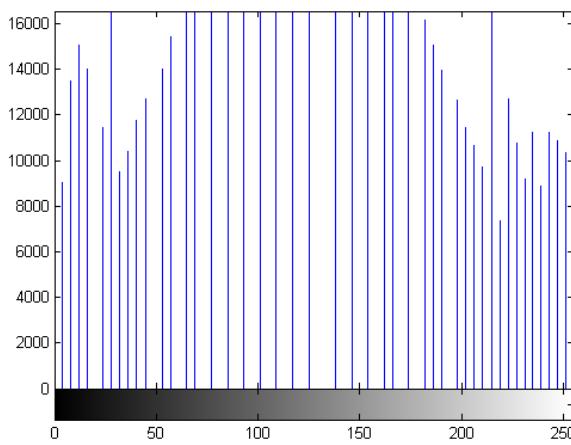


Figura f.9 - Histograma de imagen procesada (A2)
con histeq

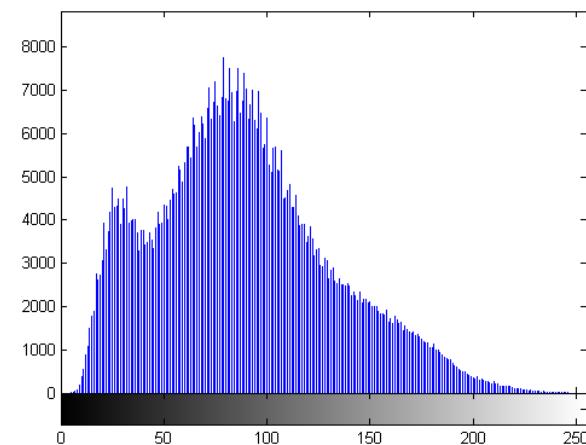


Figura f.10 - Histograma de imagen procesada (A3)
con adapthisteq

Como vemos el mejor resultado se obtiene con la función adapthisteq que consigue contrastar de forma diferente según la zona, obteniendo un histograma bastante expandido y uniforme, mientras que imadjust e histeq sobre todo, se exceden aumentando el contraste de toda la imagen sobresaturando algunas zonas, por lo que habría que hacer uso de sus diferentes parámetros de entrada para corregir la salida.

Ejemplo 3: Aplicar CLAHE a una imagen en color del tipo RGB.

Para mejorar el contraste en imágenes en color se suele trabajar en un espacio de color que tenga los valores de intensidad en una de sus componentes, como el espacio L*a*b*, donde se actúa únicamente sobre la componente de luminosidad 'L*', lo que afecta solo a la intensidad de los pixeles y preserva los colores originales. Para el espacio de trabajo RGB se podrían procesar directamente los canales de forma separada, pero cambiaría el resultado final, apareciendo más colores que en la imagen original.

```
RGB = imread('Nurnberg.tif');

% Creamos la estructura de transformación de color de sRGB a L*a*b* y
% convertimos los valores de color RGB a L*a*b* según dicha estructura.

cform2lab = makecform('srgb2lab');
LAB = applycform(RGB, cform2lab);

% Pasamos a formato de tipo double para conseguir un espacio de color
L*a*b* original del CIE y escalamos los valores de luminosidad pasando de
rango [0,100] a [0,1] que es el apropiado para imágenes de intensidad en
formato double, consiguiendo pasar a escala de grises.

LAB = lab2double(LAB);
L = LAB(:,:,1)/100;

% Aplicamos CLAHE y guardamos los valores en el lugar original de la
estructura LAB, volviéndolos a escalar al rango original [0,100].

LAB(:,:,1) = adapthisteq(L)*100;

% Volvemos a espacio de color tipo RGB y mostramos los resultados.

cform2srgb = makecform('lab2srgb');
PRGB = applycform(LAB, cform2srgb);

figure, imshow(RGB);
figure, imshow(PRGB)
```



Figura f.11 - Imagen original (RGB) sin procesar



Figura f.12 - Imagen procesada (PRGB) con
adapthisteq con los parámetros por defecto

Funciones relacionadas:

histeq, imadjust

analyze75info

Lectura de los metadatos del archivo de cabecera de un conjunto de datos en formato Analyze 7.5

Introducción

Analyze 7.5 es un producto para la visualización y análisis de imágenes biomédicas en 3-D cuyo formato de datos se creó para guardar imágenes de resonancias magnéticas (MRI). Sus archivos de salida son dos: una cabecera y una imagen, con el mismo nombre y diferentes extensiones. El archivo de cabecera (archivo.hdr) proporciona información a cerca de las dimensiones, identificación y procesado, y se lee mediante la función analyze75info. El archivo de imagen (archivo.img) guarda los datos de imagen cuyo tipo y orden se describen en el archivo de cabecera y se utiliza la función analyze75read para leer los datos.

Sintaxis

```
info = analyze75info(filename)
info = analyze75info(filename, 'ByteOrder', endian)
```

Descripción

info = analyze75info(filename) lee el archivo de cabecera del conjunto de datos en formato Analyze 7.5 que se ha especificado en la cadena filename. La función devuelve info, una estructura cuyos campos contienen información a cerca del conjunto de datos.

info = analyze75info(filename, 'ByteOrder', endian) lee el archivo de cabecera en formato Analyze 7.5 utilizando la ordenación de bits especificada en la variable endian. La ordenación de bits especificada puede tomar los siguientes valores:

Valor	Significado
'ieee-le'	Ordenación de bytes tipo <i>Little Endian</i> (guarda el byte menos significativo en la dirección más baja)
'ieee-be'	Ordenación de bytes tipo <i>Big Endian</i> (guarda el byte más significativo en la dirección más baja).

Si el valor especificado en `endian` produce un error de lectura, `analyze75info` dará un aviso y procederá a leer la cabecera con el formato de `ByteOrder` opuesto.

Ejemplos de uso

Ejemplo 1: Leer el archivo de cabecera especificando la ordenación de bytes del conjunto de datos:

```
info = analyze75info('brainMRI', 'ByteOrder', 'ieee-le');
```

Funciones relacionadas

`analyze75read`

analyze75read

Lectura de los datos de imagen de un conjunto de datos en formato Analyze 7.5

Introducción

Analyze 7.5 es un producto de software para la visualización y análisis de imágenes biomédicas en 3-D cuyo formato de datos se creó para guardar imágenes de resonancias magnéticas (MRI). Sus archivos de salida son dos: una cabecera y una imagen, con el mismo nombre y diferentes extensiones. El archivo de cabecera (archivo.hdr) proporciona información a cerca de las dimensiones, identificación y procesado y se lee mediante la función `analyze75info`. El archivo de imagen (archivo.img) guarda los datos de imagen cuyo tipo y orden se describen en el archivo de cabecera y se utiliza la función `analyze75read` para leer los datos.

Sintaxis

```
X = analyze75read(filename)  
X = analyze75read(info)
```

Descripción

`X = analyze75read(filename)` lee los datos de imagen del conjunto de datos en formato Analyze 7.5 especificado en la cadena `filename`. La función guarda los datos de la imagen en la variable `X`. Para imágenes de un solo fotograma en escala de grises, `X` es un array de `m`-por-`n`. `analyze75read` utiliza para `X` un tipo de dato que es coherente con el tipo de dato especificado en la cabecera del conjunto de datos.

`X = analyze75read(info)` lee los datos del archivo de imagen especificado en la estructura de metadatos guardada en la variable `info`, que debe ser una estructura válida devuelta por la función `analyze75info`.

`X` puede ser de tipo `logical`, `uint8`, `int16`, `int32`, `single`, o `double`. No es posible utilizar los tipos de datos complejos y RGB.

Nota: `analyze75read` devuelve los datos de imagen con orientación radiológica (LAS) que es el tipo por defecto en el formato Analyze 7.5. Por tanto, para

visualizar la imagen correctamente en MATLAB® se deben girar los datos usando el código: `x = flipdim(x,1);`

Ejemplos de uso

Ejemplo 1: Leer los datos de imagen de un archivo de imagen formato Analyze 7.5.

```
% Leemos y giramos los datos para visualizar la imagen correctamente.  
X = analyze75read('brainMRI');  
X = flipdim(X,1);  
  
% Seleccionamos todos los fotogramas en la función 'reshape' creando un  
array redimensionado adecuado para la función 'montage', que al ser  
imágenes en escala de grises, deberá tener un array de entrada de M-por-  
N-por-1-por-K (donde K es el número total de fotogramas a mostrar) para  
poder realizar correctamente el montaje de imágenes.  
  
Y = reshape(X,[size(X,1) size(X,2) 1 size(X,3)]);  
  
% Realizamos el montaje usando el mapa de color 'gray'(escala de grises).  
montage(Y, gray);
```

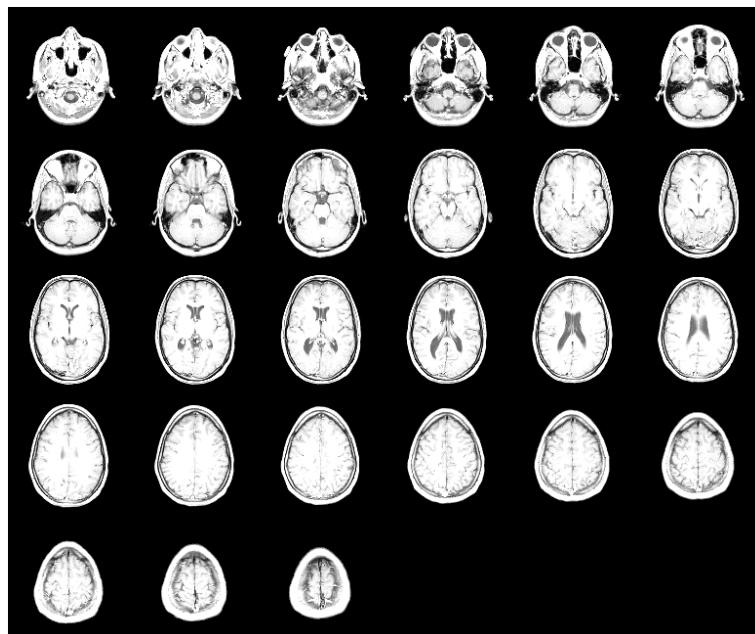


Figura f.13 - Lectura de imagen formato Analyze 7.5 y montaje

Ejemplo 2: Leer los datos de imagen a partir de la lectura de los metadatos de la cabecera.

```
info = analyze75info('brainMRI.hdr');  
X = analyze75read(info);
```

Funciones relacionadas

`analyze75info`

applycform

Transformación del espacio de color

Sintaxis

```
B = applycform(A,C)
```

Descripción

`B = applycform(A,C)` convierte los valores de color de `A` al espacio de color especificado en la estructura de transformación de color `C`. Esta estructura de transformación de color especifica varios parámetros de la transformación y se crea mediante la función `makecform`.

Si `A` es bidimensional, cada fila se interpreta como un color a menos que la estructura de transformación de color contenga un perfil ICC (*International Color Consortium*) en escala de grises, en cuyo caso, `applycform` interpreta cada pixel como un color. `B` tiene el mismo tamaño que `A`. `A` puede tener una o más columnas, dependiendo del espacio de color de entrada. `B` tiene el mismo número de filas y una o más columnas dependiendo del espacio de color de salida (las especificaciones del ICC contemplan hasta 15 espacios de color).

Si `A` es tridimensional, cada localización fila-columna se interpreta como un color y `size(A,3)` es normalmente 1 o más, dependiendo del espacio de color de entrada. `B` tiene el mismo número de filas y columnas que `A` y `size(B,3)` es 1 o más, dependiendo del espacio de color de salida.

`A` es un array real no disperso de tipo `uint8`, `uint16`, `double` o una cadena. `A` es solo una cadena si `C` se creó con la siguiente sintaxis: `C = makecform('named', profile, space)`. El array de salida `B` será de la misma clase que `A` salvo cuando el espacio de salida sea `XYZ`.

Ejemplos de uso

Ejemplo 1: Transformar el espacio de color de una imagen pasando de sRGB a $L^*a^*b^*$ creando una estructura de transformación de color previa.

```
% Lectura de una imagen en color que usa el espacio de color sRGB.  
rgb = imread('peppers.png');  
  
% Creamos la estructura de transformación de sRGB a LAB.  
C = makecform('srgb2lab');  
  
% Aplicamos la transformación.  
lab = applycform(rgb,C);
```

Funciones relacionadas

`lab2double`, `lab2uint16`, `lab2uint8`, `makecform`, `whitepoint`, `xyz2double`,
`xyz2uint16`

applylut

Operaciones de vecindad en imágenes binarias con tablas de consulta

Introducción

Ciertas operaciones con imágenes binarias se puede implementar de manera más eficiente utilizando tablas de consulta (LUT, *lookup table*), una estructura de datos utilizada para reemplazar un cálculo por una operación más simple de indexado de un array. La idea es calcular de antemano la salida para cada posible configuración de vecindad y guardar el resultado en una tabla para utilizarla posteriormente con un simple indexado. Las tablas de consulta se crean con la función `makelut` o directamente a mano si la función a evaluar es sencilla.

La función `applylut` se engloba dentro de las funciones utilizadas en el procesado morfológico de una imagen, especialmente en las técnicas de dilatación y erosión. Con esta función se puede realizar más rápido la transformación de localización llevada a cabo por la función `bwhitmiss` cuando los elementos estructurantes utilizados son pequeños.

Sintaxis

```
A = applylut(BW,LUT)
```

Descripción

`applylut` realiza una operación de vecindad en una imagen binaria produciendo primero un índice para cada vecindad de la imagen (índice único para cada posible configuración de vecindad) que permitirá localizar en la tabla de consulta `lut` el valor de salida para esa configuración de vecindad. El tipo de algoritmo utilizado depende de si se usan vecindades de 2-por-2 o 3-por-3.

`A = applylut(BW,LUT)` realiza una operación de vecindad de 2-por-2 o 3-por-3 en la imagen binaria `BW` utilizando una tabla de consulta `LUT`, que es un vector binario de 16 elementos (vecindad de 2-por-2) o de 512 elementos (vecindad de 3-por-3) devuelto por la función `makelut` y que contiene la salida (positiva o negativa) a una función determinada, para cada una de las 16 o 512 posibles vecindades binarias.

`BW` puede ser del tipo `numeric` o `logical` y debe ser real, bidimensional y no disperso. `LUT` puede ser `numeric` o `logical` y debe ser un vector real con 16 o 512 elementos. Si todos los elementos de `LUT` son 0 o 1, entonces `A` es `logical`. Si todos los elementos de `LUT` son enteros entre 0 y 255, entonces `A` es `uint8`. Para el resto de casos, `A` es `double`.

Ejemplos de uso

Ejemplo 1: Al trabajar con tablas de consulta, se le asigna un número diferente a cada configuración de vecindad posible para luego poder indexarlas fácilmente. Esto se hace multiplicando elemento a elemento cada configuración de vecindad por la matriz:

1 4
2 8

para el caso de 2-por-2

1 8 64
2 16 128
4 32 256

para el caso de 3-por-3

y sumando posteriormente todos los productos, lo que nos dará un valor entre 0 y 15 para vecindades de 2-por-2 o entre 0 y 511 para vecindades de 3-por-3.

Siguiendo este procedimiento, crear una tabla de consulta para el patrón siguiente:

0 0 0
0 1 0
1 1 1

% Lo primero es calcular el valor del índice del patrón dado: 0x1 + 0x2 + 1x4 + 0x8 + 1x16 + 1x32 + 0x64 + 0x128 + 1x256 = 4 + 16 + 32 + 256 = 308.

% Creamos la tabla de consulta poniendo a 1 el índice anterior, lo que significa que para el patrón dado, la salida es verdadera, siendo falsa o nula para el resto de configuraciones de vecindad posibles:

```
lut = zeros(512,1);  
lut(309) = 1;
```

Ejemplo 2: Realizar una transformación morfológica de erosión a una imagen (reducir el nivel de los píxeles del entorno, los objetos de tamaño menor al del objeto estructurante desaparecen) utilizando una vecindad de 2-por-2 y diferentes tablas de consulta en función de número de unos en la vecindad.

% Creamos la tabla de consulta con makelut que pasa una a una todas las vecindades de 2-por-2 a la función especificada y guarda el resultado. En este caso, la función devuelve TRUE si el número de 1's en la vecindad pasada es 4 y devuelve FALSE en caso contrario.

```
lut = makelut('sum(x(:)) == 4',2);
```

% Leemos una imagen binaria y le aplicamos diferentes erosiones.

```
BW = imread('erosion_in.png');
BW2 = applylut(BW,lut);
figure, imshow(BW); figure, imshow(BW2);
```

% Creamos otra tabla de consulta en donde la función devuelve TRUE si el número de 1's en la vecindad es 2 y devuelve FALSE en caso contrario.

```
lut = makelut('sum(x(:)) == 2',2);
BW3 = applylut(BW,lut);
figure, imshow(BW3)
```

% Creamos otra tabla de consulta en donde la función devuelve TRUE si el número de 1's en la vecindad es 1 y devuelve FALSE en caso contrario.

```
lut = makelut('sum(x(:)) == 1',2);
BW4 = applylut(BW,lut);
figure, imshow(BW4)
```

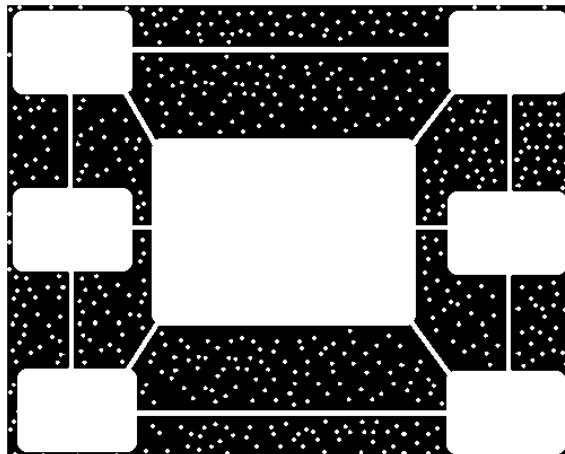


Figura f.14 - Imagen original (BW) sin procesar

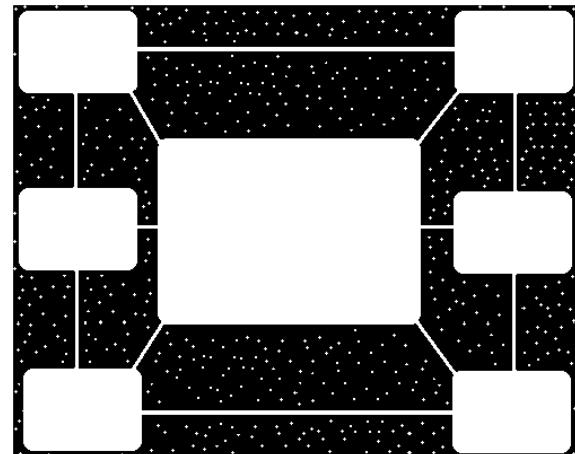


Figura f.15 - Imagen procesada (BW2) con applylut

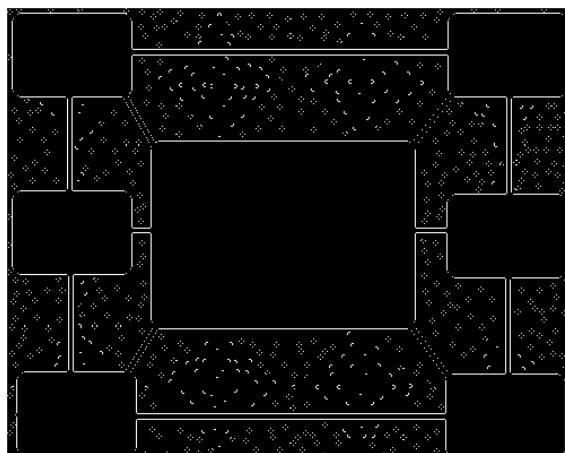


Figura f.16 - Imagen procesada (BW3) con applylut

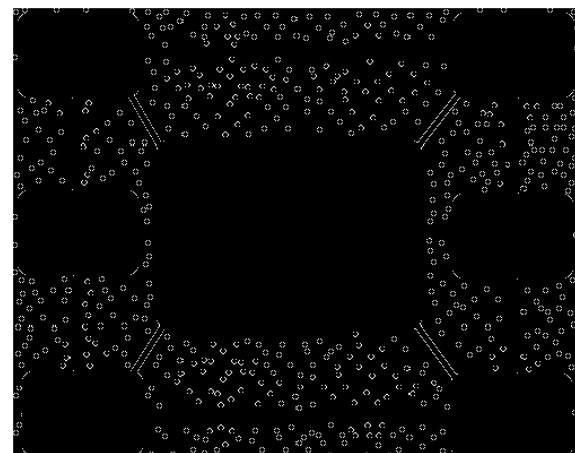


Figura f.17 - Imagen procesada (BW4) con applylut

Funciones relacionadas

bwhitmiss, makelut

axes2pix

Conversión de coordenadas en ejes a coordenadas en píxeles

Sintaxis

```
pixelx = axes2pix(dim, XDATA, AXESX)
```

Descripción

`pixelx = axes2pix(dim, XDATA, AXESX)` convierte una coordenada en ejes (como la devuelta por `get(gca, 'CurrentPoint')` por ejemplo) a una coordenada en píxeles (recomendada para localizaciones dentro de imágenes).

`AXESX` debe estar en coordenadas intrínsecas (coordenadas tales que la esquina superior izquierda de la imagen está en las coordenadas (0.5,0.5) y la esquina inferior derecha de la imagen está en (`numCols + 0.5, numRows + 0.5`). Estas coordenadas coinciden con las coordenadas en forma de fila y columna de cada pixel (índice de píxeles) si están referidas al centro de los píxeles, pero de forma inversa (el centro de la fila 2 y la columna 3 se corresponden con $x=3$ e $y=2$ en coordenadas intrínsecas).

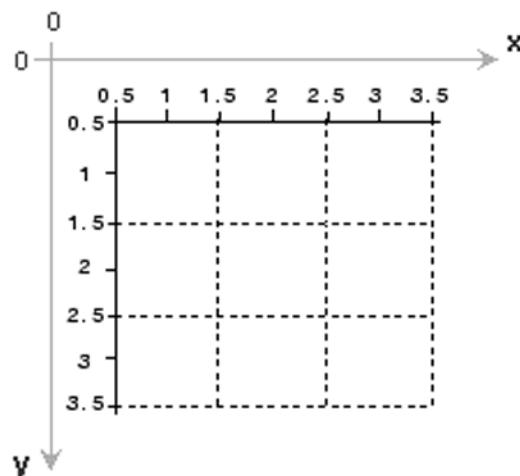


Figura f.18 - Coordenadas intrínsecas

`XDATA` es un vector de dos elementos devuelto por `get(image_handle, 'XData')` o `get(image_handle, 'YData')` que devuelven las propiedades '`XData`' e '`YData`' respectivamente. Estas propiedades controlan el sistema de coordenadas de la imagen y son por defecto [1 n] y [1 m] respectivamente para una imagen de m-por-n, lo que conlleva

a que la columna de la izquierda de la imagen tenga una coordenada $x=1$ y la fila de arriba tenga una coordenada $y=1$).

`dim` es el número de columnas que tiene la imagen para la coordenada x o el número de filas que tiene para la coordenada y .

`dim`, `XDATA`, y `AXESX` pueden ser de clase `double`. La salida es de clase `double`.

`axes2pix` realiza un chequeo mínimo de la validez de las variables `AXESX`, `DIM`, o `XDATA`. Por ejemplo, `axes2pix` devuelve una coordenada negativa si `AXESX` es menor que `XDATA(1)`. La función `axes2pix` no está diseñada para detectar errores.

Ejemplos de uso

Ejemplo 1: Pasar la coordenada (30,30) a coordenada en píxeles con los valores por defecto de XData e YData (sistema de coordenadas espaciales por defecto).

```
% Mostramos la imagen con los ejes por defecto.  
h = imshow('imagetest.tif');  
  
% Recogemos el valor de xdata e ydata por defecto.  
  
xdata = get(h,'XData');  
ydata = get(h,'YData');  
  
% Convertimos las coordenadas.  
  
[nrows,ncols] = size(get(h,'CData'));  
px = axes2pix(ncols,xdata,30)  
py = axes2pix(nrows,ydata,30)  
  
px =  
    88.1339  
  
py =  
    30
```

Ejemplo 2: Pasar la coordenada (30,30) a coordenada en píxeles con los valores no por defecto de XData e YData (sistema de coordenadas espaciales no por defecto) y comparando cómo se muestra la imagen en cada sistema de coordenadas.

```
% Mostramos la imagen con los ejes por defecto.  
h = imshow('imagetest.tif');  
  
% Definimos un valor de xdata e ydata.
```

```
xdata = [10 100];
ydata = [20 90];

% Convertimos las coordenadas.

[nrows,ncols] = size(get(h,'CData'));
px = axes2pix(ncols,xdata,30)
py = axes2pix(nrows,ydata,30)

% Mostramos la imagen en el nuevo sistema de coordenadas.

r = imread('imagetest.tif');
figure, image(r,'XData',xdata,'YData',ydata);

px =
290.1111

py =
49.7143
```

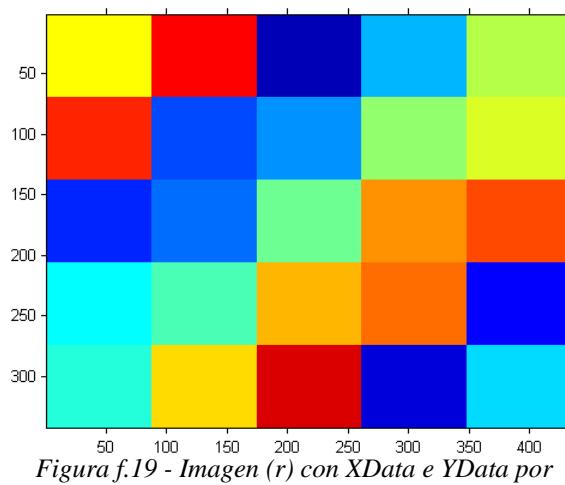


Figura f.19 - Imagen (r) con XData e YData por defecto

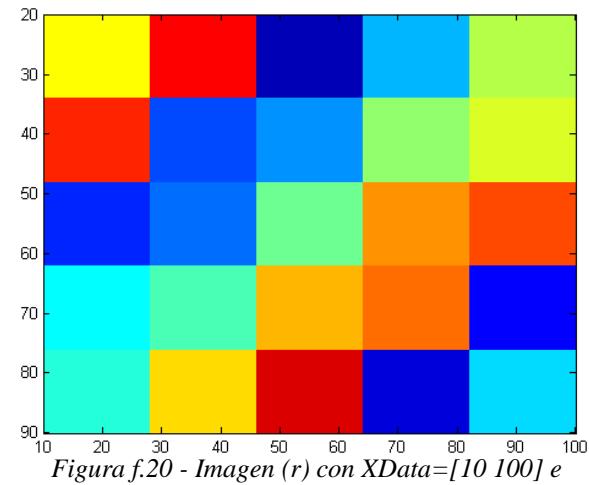


Figura f.20 - Imagen (r) con XData=[10 100] e YData=[20 90]

Funciones relacionadas

`impixelinfo, bwselect, imfill, impixel, improfile, roipoly`

bestblk

Determinación del tamaño de bloque óptimo para el procesado por bloques

Sintaxis

```
siz = bestblk([m n],k)  
[mb,nb] = bestblk([m n],k)
```

Descripción

`siz = bestblk([m n],k)` devuelve, para una imagen de m-por-n, el tamaño óptimo de bloque para el procesado por bloques. `k` es un escalar que especifica las máximas dimensiones de la fila y la columna del bloque; si se omite, se toma 100 como valor por defecto. El valor devuelto `siz` es un vector de 1-por-2 que contiene las dimensiones óptimas de la fila y la columna del bloque.

`[mb,nb] = bestblk([m n],k)` guarda las dimensiones de la fila y de la columna del bloque en `mb` y `nb` respectivamente.

Eligiendo un tamaño apropiado de bloque se puede mejorar significativamente el rendimiento del procesado por bloques.

Ejemplos de uso

Ejemplo 1: Determinar el tamaño óptimo de bloque para una imagen de 640 por 800 píxeles:

```
siz = bestblk([640 800],72)  
siz =  
     64      50
```

Funciones relacionadas

`blockproc`

blockproc

Procesado de una imagen por bloques

Introducción

El procesado de una imagen por bloques es una técnica en la que la imagen se procesa en bloques o regiones en vez de hacerse de una sola vez. La operación se aplica a cada bloque por separado y finalmente se unen todos los bloques para formar la imagen de salida. Es útil normalmente para el procesado de imágenes muy grandes.

Sintaxis

```
B = blockproc(A,[M N],fun)
B = blockproc(src_filename,[M N],fun)
B = blockproc(adapter,[M N],fun)
blockproc(...,param,val,...)
```

Descripción

`B = blockproc(A,[M N],fun)` procesa la imagen `A` aplicando la función `fun` a cada bloque distinto de `M`-por-`N` de `A`, concatenando los resultados en la matriz de salida `B`. Estas secciones o bloques diferentes, cubren la imagen empezando por la esquina superior izquierda, sin solaparse. Si los bloques no encajan perfectamente encima de la imagen, se puede añadir un relleno a la imagen o trabajar con bloques parciales en los bordes derecho o inferior de la imagen (por defecto).

`fun` es una referencia o puntero a una función (*function handle*) que acepta una estructura de bloque (*block struct*) como entrada y devuelve una matriz, un vector o un escalar `y`. Por ejemplo, `y = fun(block_struct)`. `blockproc` pasa cada bloque de dato de la imagen de entrada `A` a la función `fun`, que produce `y`, el bloque correspondiente en la imagen de salida. Si `y` está vacío, `blockproc` no genera ninguna salida y vuelve vacía después de procesar todos los bloques. Eligiendo un tamaño apropiado de bloque se puede mejorar significativamente el rendimiento. Más detalles sobre las estructuras de bloque (*block struct*) al final de la definición de parámetros.

`B = blockproc(src_filename,[M N],fun)` procesa la imagen `src_filename`, leyendo y procesando un bloque cada vez en lugar de leerla toda primero con `imread`. Esta sintaxis

es apropiada para procesar imágenes muy grandes ya que así solo se almacena un bloque de imagen en memoria cada vez. Si la matriz de salida `B` es demasiado grande para almacenarse en memoria se debe omitir el guardar la salida en esa variable y escribir cada bloque directamente en un archivo, especificándolo con el parámetro '`Destination`'.

`B = blockproc(adapter,[M N],fun)` procesa la imagen especificada por `adapter`, que es un objeto `ImageAdapter` (clase definida por el usuario que le proporciona a `blockproc` una API para la lectura y escritura de un formato específico de imagen).

`blockproc(...,param,val,...)` procesa la imagen de entrada teniendo en cuenta determinados parámetros que controlan varios aspectos del comportamiento de los bloques. Los parámetros no son sensibles a las mayúsculas:

Parámetro	Valor
<code>'BorderSize'</code>	<p>Vector de dos elementos <code>[v h]</code> que especifica la cantidad de píxeles de borde a añadir a cada bloque, borde que se eliminará por defecto del resultado de <code>fun</code> (ver parámetro '<code>TrimBorder</code>') y que por tanto no aparecerá en la composición final. La función lo añade <code>v</code> filas encima y debajo de cada bloque y <code>h</code> columnas a la izquierda y derecha de cada bloque. El tamaño final de cada bloque será:</p> $[M + 2*v, N + 2*h]$ <p>El valor por defecto es <code>[0 0]</code>, que significa sin borde. Por defecto, la función elimina el borde automáticamente del resultado de <code>fun</code>. Consultar el parámetro '<code>TrimBorder</code>' para más información. La función rellena los bloques con bordes usando ceros.</p> <p>(Por defecto: [0 0] (sin borde))</p>
<code>'Destination'</code>	<p>Especifica el destino de la salida de <code>blockproc</code>. Cuando se especifica, <code>blockproc</code> no devuelve la imagen procesada como un argumento de salida sino que escribe la salida en ese destino especificado. Los valores posibles de '<code>Destination</code>' son:</p> <ul style="list-style-type: none"> Nombre de archivo en formato TIFF: una cadena acabada en la extensión <code>'.tif'</code>. Si la imagen ya existe, se sobrescribirá. Objeto <code>ImageAdapter</code>: una instancia de una clase <code>ImageAdapter</code> que permite la lectura y escritura en archivos de imagen no comunes. Más información en la página dedicada a <code>ImageAdapter</code> (p. 333) <p>El parámetro '<code>Destination</code>' es útil cuando esperas que la salida</p>

	sea demasiado grande para caber en memoria.
'PadPartialBlocks'	<p>Un escalar tipo logical.</p> <p>Cuando tiene valor <code>true</code>, <code>blockproc</code> rellena los bloques parciales para hacerlos tamaño completo (M-por-N). Si hay bloques parciales estarán en el borde derecho e inferior de la imagen. <code>blockproc</code> usa ceros para llenar los bloques parciales.</p> <p>Por defecto es <code>false</code>, lo que significa que no se llenan los bloques parciales y se procesan con el tamaño especificado.</p> <p>(Por defecto: <code>false</code>)</p>
'PadMethod'	<p>El método de relleno o '<code>PadMethod</code>' determina cómo <code>blockproc</code> llenará las fronteras entre bloques. Estas son las opciones:</p> <ul style="list-style-type: none"> • <code>x</code>: Rellena con un valor de relleno escalar (<code>X</code>). (Por defecto: <code>x == 0</code>) • '<code>replicate</code>': Repite los bordes de la imagen de entrada. • '<code>symmetric</code>': Rellena la imagen con reflejos de sí misma.
'TrimBorder'	<p>Un escalar tipo logical.</p> <p>Cuando tiene valor <code>true</code>, la función <code>blockproc</code> recorta píxeles del borde en la salida de la función de usuario, <code>fun</code>. La función elimina v filas de la parte superior e inferior de la salida de <code>fun</code> y h columnas de los bordes izquierdo y derecho. v y h los define el parámetro '<code>BorderSize</code>'.</p> <p>El valor por defecto es <code>true</code>, lo que significa que la función elimina automáticamente los bordes de la salida de <code>fun</code>.</p> <p>(Por defecto: <code>true</code>)</p>
'UseParallel'	<p>Un escalar tipo logical.</p> <p>Cuando tiene valor <code>true</code>, la función <code>blockproc</code> intentará trabajar en modo paralelo, distribuyendo el procesado a través de diferentes sesiones de trabajo MATLAB si se cuenta con la <i>toolbox</i> para el cálculo en paralelo. Cuando se trabaja en modo paralelo, la imagen de entrada ya no puede ser un objeto <code>ImageAdapter</code>.</p>

Los archivos de entrada y salida de `blockproc` (especificados en los parámetros `src_filename` y '`Destination`') deben tener las siguientes extensiones:

- Formatos de lectura/escritura: TIFF (*.tif, *.tiff), JPEG2000 (*.jp2, *.j2c, *.j2k)
- Formatos de solo lectura: JPEG2000 (*.jpg, *.jpx)

Estructuras de bloque (*block struct*)

Una estructura de bloque en MATLAB contiene los datos del bloque así como otra información relativa. Estos son los campos de la estructura:

- `block_struct.border`: vector de dos elementos [v h] que especifica el tamaño del relleno vertical y horizontal alrededor del bloque de datos. Relacionado con el parámetro de entrada '`BorderSize`'.
- `block_struct.blockSize`: vector de dos elementos [rows cols], que especifica el tamaño del bloque de datos. Si se ha definido un borde su tamaño no se cuenta.
- `block_struct.data`: matriz de M-por-N o M-por-N-por-P con los datos del bloque.
- `block_struct.imageSize`: vector de dos elementos [rows cols], que especifica el tamaño completo de la imagen de entrada.
- `block_struct.location`: vector de dos elementos [row col], que especifica la posición del primer pixel (menor-fila, menor-columna) de los datos del bloque en la imagen de entrada. Si se ha definido un borde, se refiere al primer pixel sin borde.

Ejemplos de uso

Ejemplo 1: Crear una miniatura de una imagen grande utilizando procesado por bloques, sin tener que leer la imagen entera en memoria.

Utilizar procesado por bloques para redimensionar una imagen puede que no produzca los mismos resultados que redimensionarla toda de una vez (`I2=imresize(I,0.7)`) debido a los efectos de los bordes en los bloques.

```
% Guardamos la función "redimensionar estructura de datos a escala 0.7" en la variable fun.
```

```
fun = @(block_struct) imresize(block_struct.data,0.7);

% Procesamos la imagen por bloques con la función especificada, usando un tamaño de bloque de 100 por 100 y guardando cada bloque en la imagen final 'papples.tif'.

blockproc('apples.tif',[100 100],fun,'Destination','papples.tif');
```



Figura f.21 - Imagen original (apples.tif) sin procesar



Figura f.22 - Imagen redimensionada (papples.tif) por bloques con blockproc

Ejemplo 2: Intercambiar los canales de color rojo y verde de una imagen RGB y escribir el resultado en una nueva imagen utilizando procesando por bloques.

```
I = imread('apples.tif');
imshow(I)

% Creamos la variable función para intercambiar los canales.

fun = @(block_struct) block_struct.data(:, :, [2 1 3]);

% Procesamos por bloques. Se podría hacer directamente sin bloques usando
el código: I2=I(:, :, [2 1 3]).

blockproc(I,[200 200],fun,'Destination','grb.tif');
figure, imshow('grb.tif')
```



Figura f.21 - Imagen original (I) sin procesar



Figura f.23 - Imagen procesada (grb.tif) por bloques con blockproc intercambiando sus canales

Ejemplo 3: Aplicar un filtro Gaussiano a una imagen utilizando procesado por bloques.

```
% Leemos una imagen de 512 por 512 píxeles.
```

```
I = imread('apples512.tif');
imshow(I)
```

```
% Creamos un filtro pasabajo Gaussiano de 5 filas por 5 columnas.  
h = fspecial('gaussian',5,2);  
  
% Creamos la función que procesará (filtrará) cada bloque.  
fun = @(block_struct) imfilter(block_struct.data,h);  
  
% Procesamos con bloques de 128 por 128 píxeles que llenan la imagen sin  
necesitar bloques parciales y mostramos el resultado. Aparecerán bordes  
negros indeseables no evitables en cada bloque, creados por la función de  
filtrado imfilter para llenar cada bloque.  
  
P1 = blockproc(I,[128 128],fun);  
figure, imshow(P1)  
  
% Procesamos igual que antes pero especificando un borde de 2 píxeles en  
cada bloque y que no se elimine el borde al componer la imagen final,  
para ver cómo estos bordes caen exactamente encima de los indeseados que  
generará el filtrado y se evita así la pérdida de información. El tamaño  
final aumenta pero los bordes no eliminan información de la imagen y  
están listos para ser eliminados.  
  
P2 = blockproc(I,[128 128],fun,'BorderSize',[2 2],'TrimBorder',false);  
figure, imshow(P2)  
  
% Procesamos de nuevo pero especificando que sí se borren los bordes  
superpuestos para dejar la imagen procesada sin bordes y sin pérdidas.  
  
P3 = blockproc(I,[128 128],fun,'BorderSize',[2 2]);  
figure, imshow(P3)
```



Figura f.24 - Imagen original (I) sin procesar

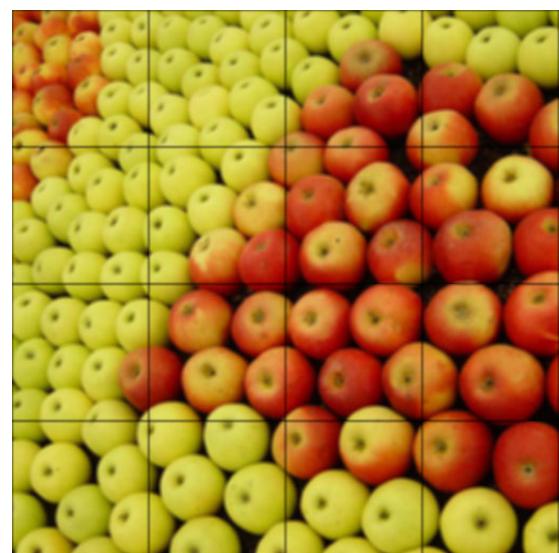


Figura f.25 - Imagen procesada (P1) por bloques con
blockproc con filtro Gaussiano y bordes indeseados
creados por el filtrado de cada bloque

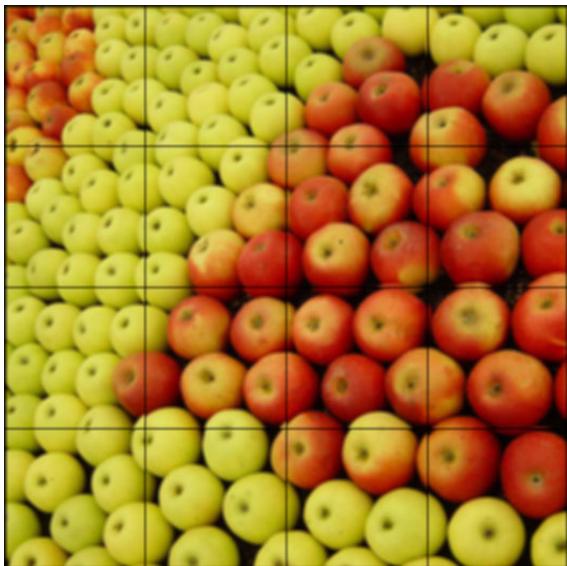


Figura f.26 - Imagen procesada (P2) por bloques con blockproc con filtro Gaussiano, usando bordes añadidos no eliminados que caen encima de los que crearán el filtrado y evitan así la pérdida de información



Figura f.27 - Imagen procesada (P3) por bloques con blockproc, usando bordes añadidos y eliminados

Ejemplo 4: Comprimir una imagen en escala de grises utilizando la DCT (Transformada de coseno discreta) mediante procesado por bloques.

```
% Leemos imagen original, su tamaño y la mostramos.

I = imread('rueda.jpg');
DIR = dir('rueda.jpg');
tamorig=DIR.bytes
imshow(I)

I2=im2double(I);

% Matriz DCT de 8x8 en formato double.

T=dctmtx(8);

% Procesado de la imagen en bloques de 8x8 (cada bloque sufre la transformada DCT matricial sin pérdida de información).

B1 = blkproc(I2,[8 8],'P1*x*P2',T,T');

% Definimos la máscara que eliminará redundancia (compresión).
% Elimina las frecuencias altas, en donde suele estar la redundancia.

mask = [1 1 1 0 0 0 0 0
        1 0 0 0 0 0 0 0
        1 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0];

% Uso la máscara para eliminar redundancia.
```

```
B2=blkproc(B1,[8 8],'P1.*x',mask);  
  
% Hacemos la DCT inversa para ver el resultado.  
  
I3=blkproc(B2,[8 8],'P1*x*P2',T',T);  
  
% Guardamos imagen comprimida y leemos su tamaño.  
  
figure, imshow(I3)  
imwrite(I3, 'prueda.jpg');  
DIR = dir('prueda.jpg');  
tamproc=DIR.bytes  
  
tamorig =  
513288  
  
tamproc =  
50984
```



Figura f.28 - Imagen original (*I*) sin procesar



Figura f.29 - Imagen comprimida (*I3*) por bloques con *blockproc*

Funciones relacionadas

colfilt, function_handle, ImageAdapter, nlfilter

bwarea

Área de objetos activos en una imagen binaria

Sintaxis

```
total = bwarea(BW)
```

Descripción

total = bwarea(BW) estima el área en píxeles de los objetos activos presentes en la imagen binaria BW. total es un escalar cuyo valor corresponde aproximadamente al número total de píxeles activos en la imagen, pero puede no ser exacto ya que cada patrón de píxeles se cuenta de forma diferente. Para pasar de área en píxeles a área en unidades métricas solo debemos conocer el área del pixel en unidades métricas y multiplicar por el número total de píxeles.

BW puede ser del tipo numeric o logical. Para entradas numéricas, cualquier pixel distinto de cero se considera activo (blanco, 1). El valor de salida devuelto (total) es de tipo double.

Ejemplos de uso

Ejemplo 1: Calcular el área de los objetos activos de dos imágenes binarias de 256-por-256 píxeles.

```
BW = imread('area.tif');
BW2 = imread('area2.tif');

figure, imshow(BW)
totalp = bwarea(BW)

figure, imshow(BW2)
totalp2 = bwarea(BW2)

totalp =
    1.1281e+005

totalp2 =
    2.2561e+005
```

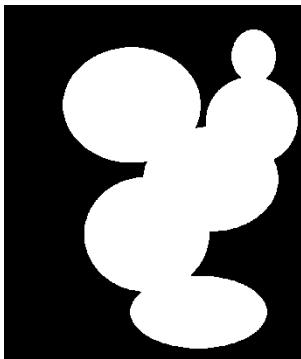


Figura f.30 - Imagen (BW) con objeto activo

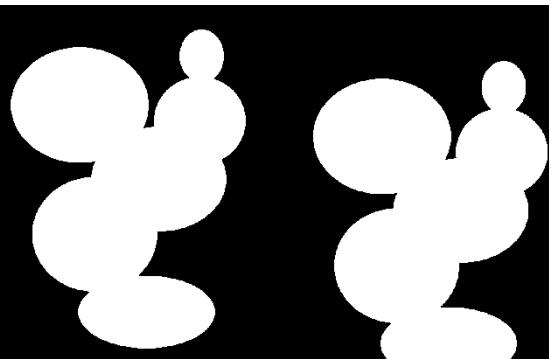


Figura f.31 - Imagen (BW2) con doble objeto activo

Funciones relacionadas

`bweuler`, `bwperim`

bwareaopen

Borrado de objetos conectados en imagen binaria

Sintaxis

```
BW2 = bwareaopen(BW, P)
BW2 = bwareaopen(BW, P, conn)
```

Descripción

`BW2 = bwareaopen(BW, P)` elimina de una imagen binaria todos los objetos conectados que tengan menos que `P` píxeles, produciendo otra imagen binaria `BW2`. La conectividad por defecto es de 8 píxeles para dos dimensiones, 26 para tres dimensiones y `conndef(ndims(BW), 'maximal')` para mayores dimensiones.

`BW2 = bwareaopen(BW, P, conn)` especifica la conectividad deseada en la variable `conn`, que puede tener los siguientes valores escalares (siempre simétrica respecto su elemento central):

Valor	Significado
Conectividades Bidimensionales	
4	Vecindad de 4 píxeles: Un par de píxeles colindantes forman parte del mismo objeto solo si están activos y conectados en vertical u horizontal.
8	Vecindad de 8 píxeles: Un par de píxeles colindantes forman parte del mismo objeto solo si están activos. (Por defecto)
Conectividades Tridimensionales	
6	Vecindad de 6 píxeles.
18	Vecindad de 18 píxeles.
26	Vecindad de 26 píxeles. (Por defecto)
<p>Nota: La conectividad se puede definir para cualquier dimensión de forma más general utilizando para <code>conn</code> una matriz de 3-por-3-por-...-por-3 de ceros y unos. Los unos definen las localizaciones de los píxeles vecinos relativos al elemento central de <code>conn</code>. Por ejemplo, <code>conn=ones(3)</code> define una conectividad de 8 píxeles.</p>	

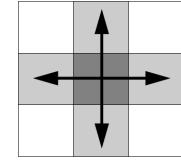


Figura f.32 - vecindad 4

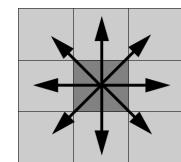


Figura f.33 - vecindad 8

BW puede ser un array del tipo numeric o logical de cualquier dimensión y debe ser no disperso. BW2 , el valor devuelto, es de tipo logical.

Ejemplos de uso

Ejemplo 1: Borrar de la imagen binaria `text.png` todos los objetos conectados (con vecindad de 8) que contengan menos de 700, 1000 y 1200 píxeles.

```
BW = imread('text.tif');
BW2 = bwareaopen(BW, 700);
BW3 = bwareaopen(BW, 1000);
BW4 = bwareaopen(BW, 1200);
figure, imshow(BW)
figure, imshow(BW2)
figure, imshow(BW3)
figure, imshow(BW4)
```



Figura f.34 - Imagen original (BW) sin procesar

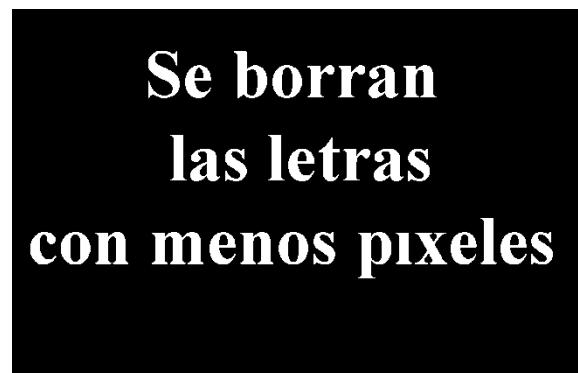


Figura f.35 - Imagen procesada (BW2) con
bwareaopen con $P=700$ píxeles mínimos

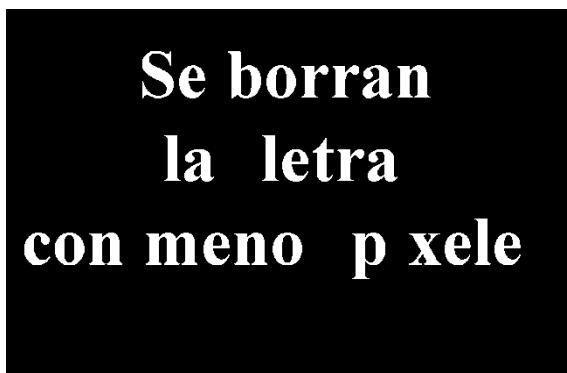


Figura f.36 - Imagen procesada (BW3) con
bwareaopen con $P=1000$ píxeles mínimos

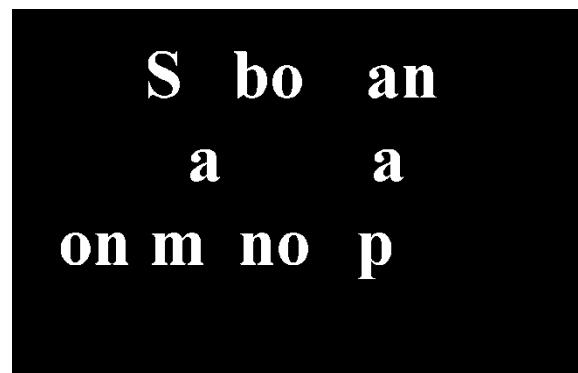


Figura f.37 - Imagen procesada (BW4) con
bwareaopen con $P=1200$ píxeles mínimos

Funciones relacionadas

`bwconncomp`, `conndef`

bwboundaries

Cálculo de las fronteras de los objetos de una imagen binaria

Sintaxis

```
B = bwboundaries(BW)
B = bwboundaries(BW,conn)
B = bwboundaries(BW,conn,options)
[B,L] = bwboundaries(...)
[B,L,N,A] = bwboundaries(...)
```

Descripción

`B = bwboundaries(BW)` calcula las fronteras exteriores de los objetos de la imagen binaria `BW`, así como las fronteras de los huecos que están dentro de los objetos. `bwboundaries` además calcula las fronteras de los ‘objetos hijos’ que están dentro de los ‘objetos padre’. A diferencia de la función `bwtraceboundary`, que solo calcula la frontera de un objeto, `bwboundaries` calcula todas las fronteras de todos los objetos y huecos.

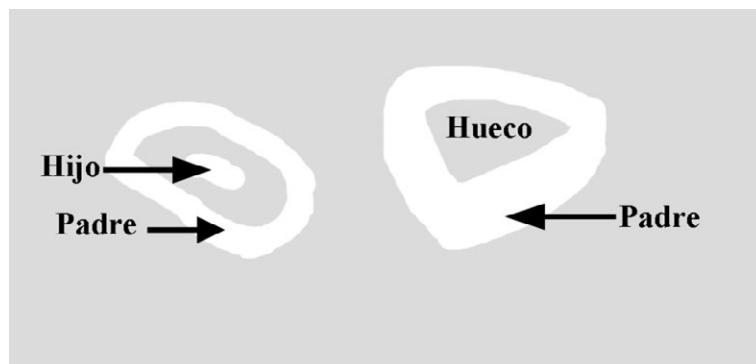


Figura f.38 - Objetos padre e hijo

`BW` debe ser una imagen binaria donde los píxeles distintos de cero sean los objetos y los píxeles con valor 0 sean el fondo de la imagen.

`bwboundaries` devuelve `B`, una matriz de P -por-1 celdas donde P es el número de objetos y huecos. Cada celda en la matriz contiene una matriz de Q -por-2 y cada fila de la matriz contiene las coordenadas en fila y columna de un pixel frontera. Q es el número de píxeles frontera para la región correspondiente.

`B = bwboundaries(BW,conn)` especifica la conectividad a usar cuando se trazan fronteras padre-hijo. `conn` puede tener los siguientes valores escalares:

Valor	Significado
4	Vecindad de 4 píxeles.
8	Vecindad de 8 píxeles. (Por defecto)

`B = bwboundaries(BW,conn,options)` especifica un argumento opcional en el que la variable `options` puede tomar los siguientes valores:

Valor	Significado
'holes'	Busca fronteras de tanto objetos como huecos. (Por defecto)
'noholes'	Busca solo fronteras de objetos (padres e hijos). Puede mejorar el rendimiento.

`[B,L] = bwboundaries(...)` devuelve la matriz de etiquetas `L` como segundo parámetro de salida, en la que están etiquetados los objetos y los huecos. `L` es un array bidimensional de enteros no negativos que representa regiones contiguas. La región número `k` incluye todos los elementos en `L` que tienen el valor `k`. El número de objetos y huecos representado por `L` es igual a `max(L(:))`. Los valores que valen 0 en `L` forman el fondo de la imagen.

`[B,L,N,A] = bwboundaries(...)` devuelve `N`, el número de objetos encontrados, y `A`, una matriz de adyacencia (matriz cuadrada que se utiliza para representar relaciones binarias). Las primeras `N` celdas de `B` son fronteras de objetos. `A` representa las dependencias padre-hijo-hueco. `A` es una matriz cuadrada, tipo `logical`, con lado de longitud `max(L(:))` y cuyas filas y columnas corresponden a las posiciones de las fronteras guardadas en `B`.

Las fronteras encerradas por `B{m}` y las fronteras que cubren a `B{m}` se pueden encontrar utilizando `A` de la siguiente manera:

```
fronteras_que_cubren = find(A(m,:));
fronteras_encerradas = find(A(:,m));
```

`BW` puede ser del tipo `numeric` o `logical` y debe ser real, bidimensional y no disperso. `L` y `N` son `double`. `A` es `logical` disperso.

Ejemplos de uso

Ejemplo 1: Mostrar las fronteras de los objetos de una imagen binaria.

```
% Leemos imagen, etiquetamos objetos y huecos y creamos fronteras.

BW = imread('original.tif');
[B,L] = bwboundaries(BW);

% Mostramos objetos etiquetados en L usando mapa de color 'jet' con fondo
% negro y fronteras en blanco.

imshow(label2rgb(L, @jet, [.0 .0 .0]))

% Recorremos el array de los píxeles frontera de cada objeto y hueco y
% trazamos las fronteras.

hold on
for k = 1:length(B)
    boundary = B{k};
    plot(boundary(:,2), boundary(:,1), 'w', 'LineWidth', 2)
end

% Repetimos pero ahora solo con objetos, no con huecos.

[B,L] = bwboundaries(BW,'noholes');
figure, imshow(label2rgb(L, @jet, [.0 .0 .0]))
hold on
for k = 1:length(B)
    boundary = B{k};
    plot(boundary(:,2), boundary(:,1), 'w', 'LineWidth', 2)
end
```



Figura f.39 - Imagen original (BW) sin procesar



Figura f.40 - Imagen procesada con bwboundaries para sacar fronteras de objetos y huecos

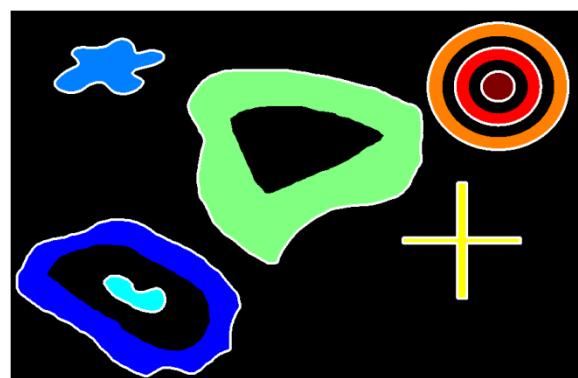


Figura f.41 - Imagen procesada con bwboundaries para sacar fronteras de objetos

Ejemplo 2: Mostrar las fronteras de los objetos de una imagen binaria. Pintar las fronteras de los objetos en rojo y de los huecos en verde.

```
BW = imread('original.tif');
[B,L,N] = bwboundaries(BW);

figure; imshow(BW); hold on;
for k=1:length(B),
    boundary = B{k};
    if(k > N)
        plot(boundary(:,2),...
              boundary(:,1),'g','LineWidth',2);
    else
        plot(boundary(:,2),...
              boundary(:,1),'r','LineWidth',2);
    end
end
```

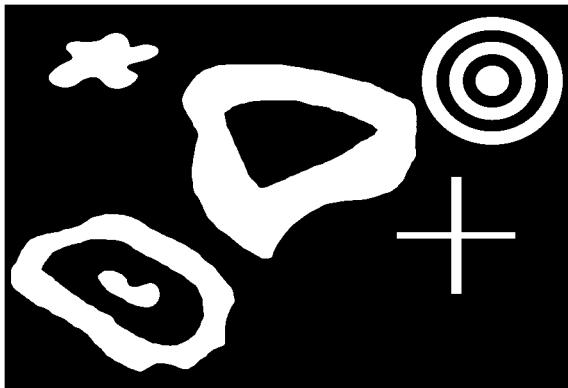


Figura f.39 - Imagen original (BW) sin procesar



Figura f.42 - Imagen procesada con bwboundaries para sacar fronteras de objetos y huecos

Ejemplo 3: Mostrar las fronteras de los objetos padre en rojo (cualquier fila vacía de la matriz de adyacencias pertenece a un objeto padre) y sus huecos en verde.

```
BW = imread('original.tif');
[B,L,N,A] = bwboundaries(BW);

figure; imshow(BW); hold on;
for k=1:length(B),
    if(~sum(A(k,:)))
        boundary = B{k};
        plot(boundary(:,2),...
              boundary(:,1),'r','LineWidth',2);
        for l=find(A(:,k))'
            boundary = B{l};
            plot(boundary(:,2),...
                  boundary(:,1),'g','LineWidth',2);
        end
    end
end
```



Figura f.39 - Imagen original (BW) sin procesar



Figura f.43 - Imagen procesada con bwboundaries para sacar fronteras de objetos y huecos

Ejemplo 4: Leer y mostrar una imagen binaria. Superponer las fronteras de la imagen. Mostrar un texto indicando el número de región (basado en la matriz de etiquetas) al lado de cada frontera. Adicionalmente, representar la matriz de adyacencias utilizando la función spy.

```
BW = imread('original.tif');
[B,L,N,A] = bwboundaries(BW);

figure, imshow(BW); hold on;
colors=['b' 'g' 'r' 'c' 'm' 'y'];

% Recorremos objetos y huecos.

for k=1:length(B)
    boundary = B{k};
    cidx = mod(k,length(colors))+1;
    plot(boundary(:,2), boundary(:,1),colors(cidx),'LineWidth',2);

    % Aleatorizamos la posición del texto para una mejor visibilidad.

    rndRow = ceil(length(boundary)/(mod(rand*k,7)+1));
    col = boundary(rndRow,2);
    row = boundary(rndRow,1);
    h = text(col+1, row-1, num2str(L(row,col)));
    set(h,'Color',colors(cidx),'FontSize',14,'FontWeight','bold');
end
figure; spy(A);
```



Figura f.39 - Imagen original (BW) sin procesar



Figura f.44 - Imagen procesada con `bwboundaries` para sacar fronteras de objetos y huecos

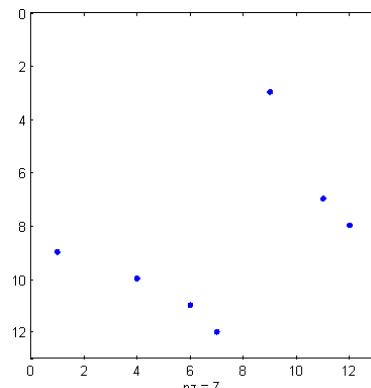


Figura f.45 - Imagen que representa la matriz de adyacencia de la imagen original (BW)

Funciones relacionadas

`bwlabel`, `bwlabeln`, `bwperim`, `bwtraceboundary`

bwconncomp

Búsqueda de componentes conectadas en una imagen binaria

Introducción

Las funciones `bwlabel`, `bwlabeln` y `bwconncomp` calculan las componentes conectadas en imágenes binarias. `bwconncomp` reemplaza el uso de `bwlabel` y `bwlabeln` usando mucha menos memoria y es normalmente más rápida que las otras funciones. Esta es una comparativa de las funciones:

Función	Dimensión de entrada	Tipo de salida	Uso de memoria	Conectividad
<code>bwlabel</code>	2-D	Matriz de etiquetas con precisión doble.	Alto	4 o 8
<code>bwlabeln</code>	N-D	Matriz de etiquetas con precisión doble.	Alto	Cualquiera
<code>bwconncomp</code>	N-D	Estructura CC.	Bajo	Cualquiera

Sintaxis

```
CC = bwconncomp(BW)
CC = bwconncomp(BW, conn)
```

Descripción

`CC = bwconncomp(BW)` devuelve las componentes conectadas `CC` encontradas en `BW`. La imagen binaria `BW` puede tener cualquier dimensión y puede ser un array del tipo `numeric` o `logical`, siempre real y no disperso. `CC` es una estructura con cuatro campos:

Campo	Descripción
<code>Connectivity</code>	Conectividad de las componentes conectadas (objetos).
<code>ImageSize</code>	Tamaño de <code>BW</code> .
<code>NumObjects</code>	Número de componentes conectadas (objetos) en <code>BW</code> .
<code>PixelIdxList</code>	Matriz de 1-por- <code>NumObjects</code> celdas donde el elemento número <code>k</code> de la matriz es un vector que contiene los índices lineales de los píxeles en el objeto número <code>k</code> .

`bwconncomp` usa por defecto la conectividad de 8 píxeles vecinos para 2 dimensiones, 26 para 3 dimensiones, y `conndef(ndims(BW), 'maximal')` para dimensiones mayores.

`CC = bwconncomp(BW, conn)` especifica la conectividad deseada para las componentes conectadas. `conn` puede tener los siguientes valores escalares (siempre simétrica respecto su elemento central):

Valor	Significado
Conectividades Bidimensionales	
4	Vecindad de 4 píxeles: Un par de píxeles colindantes forman parte del mismo objeto solo si están activos y conectados en vertical u horizontal.
8	Vecindad de 8 píxeles: Un par de píxeles colindantes forman parte del mismo objeto solo si están activos. (Por defecto)
Conectividades Tridimensionales	
6	Vecindad de 6 píxeles.
18	Vecindad de 18 píxeles.
26	Vecindad de 26 píxeles. (Por defecto)
<p>Nota: La conectividad se puede definir para cualquier dimensión de forma más general utilizando para <code>conn</code> una matriz de 3-por-3-por-...-por-3 de ceros y unos. Los unos definen las localizaciones de los píxeles vecinos relativos al elemento central de <code>conn</code>. Por ejemplo, <code>conn=ones(3)</code> define una conectividad de 8 píxeles.</p>	

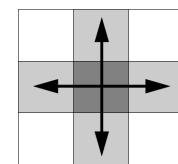


Figura f.32 - vecindad 4

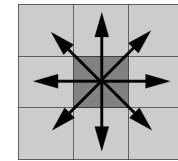


Figura f.33 - vecindad 8

Para medir las características de los objetos dentro de una imagen binaria con la función `regionprops` con conectividad por defecto se debe usar `BW` directamente en `regionprops`.

Ejemplos de uso

Ejemplo 1: Obtener las componentes conectadas de una imagen binaria y mostrar uno de los campos de la estructura de información devuelta.

```
BW = [1 1 1 0 0 0 0
      1 1 1 0 1 1 0 0
      1 1 1 0 1 1 0 0
      0 0 0 0 0 0 1 0]
```

```
0      0      0      0      0      0      1      0
0      0      0      0      0      0      1      0
0      1      1      0      0      1      1      0
0      1      1      0      0      0      0      0 ];

s = bwconncomp(BW)

s =
    Connectivity: 8
    ImageSize: [8 8]
    NumObjects: 3
    PixelIdxList: {[9x1 double]  [4x1 double]  [9x1 double]}

% El parámetro más relevante es PixelIdxList, que contiene los índices
% lineales de los píxeles pertenecientes a cada objeto. El segundo objeto
% encontrado consta de 4 píxeles con los siguientes índices lineales:

s.PixelIdxList{2}

ans =
15
16
23
24
```

Ejemplo 2: Obtener las componentes conectadas de una imagen binaria y borrar el objeto con más píxeles.

```
BW = imread('text.tif'); imshow(BW)

s = bwconncomp(BW);

% Aplicamos la función 'numel' a cada celda para saber el número de
% elementos o píxeles de cada objeto.

numPixels = cellfun(@numel,s.PixelIdxList);

% Encontramos el objeto con mayor número de píxeles y su índice.

[biggest,idx] = max(numPixels);

% Eliminamos ese objeto y mostramos la imagen.

BW(s.PixelIdxList{idx}) = 0;
figure, imshow(BW)
```



Figura f.46 - Imagen original (BW) sin procesar

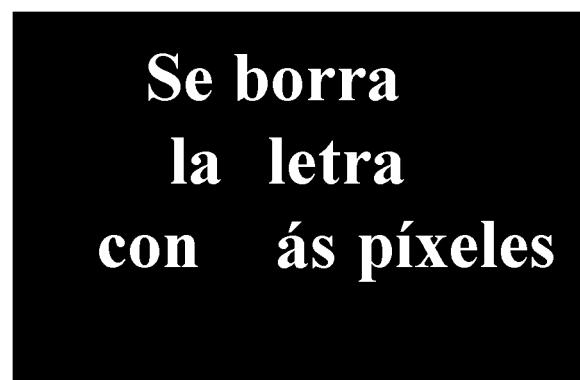


Figura f.47 - Imagen BW sin el objeto mayor

Ejemplo 3: Obtener las componentes conectadas de una imagen binaria y etiquetarlas para mostrarlas.

```
BW = imread('circulos.tif');
imshow(BW)
c = bwconncomp(BW)

% Etiquetamos las componentes con la función labelmatrix.

L = labelmatrix(c);
rgb = label2rgb(L, 'jet', [.7 .7 .7], 'shuffle');
figure, imshow(rgb)

c =
    Connectivity: 8
    ImageSize: [587 886]
    NumObjects: 10
    PixelIdxList: {1x10 cell}
```

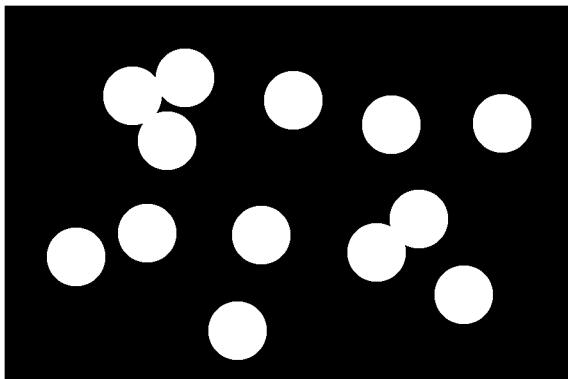


Figura f.48 - Imagen original (BW) sin procesar

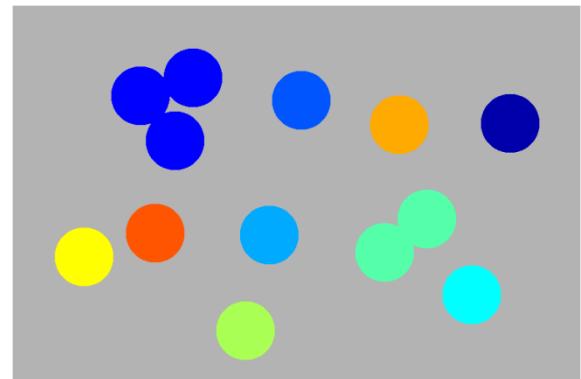


Figura f.49 - Imagen procesada (BW) con bwconncomp para encontrar objetos

Funciones relacionadas

bwlabel, bwlabeln, labelmatrix, regionprops

bwdist

Transformada de distancia (medida de separación de píxeles) de una imagen binaria

Introducción

La transformada de distancia de una imagen binaria tiene una interpretación muy sencilla: es la distancia de cada pixel al pixel más cercano distinto de cero. Se utiliza sobre todo en operaciones morfológicas como la erosión, dilatación y segmentación.

Habitualmente se utiliza en segmentación en conjunto con la transformada divisoria (función `watershed`) que es una función muy útil para dividir formas convexas que se están tocando y poder así contabilizarlas por separado. La transformada divisoria interpreta la imagen como un relieve topográfico, con los niveles de gris como alturas (más blanco más alto y más negro más hondo o más valle), y calcula las líneas divisorias, que son los límites de las cuencas hidrográficas o valles del relieve. De esta forma, si somos capaces de procesar la imagen original para que los objetos que queremos segmentar coincidan con las cuencas o valles del relieve topográfico (habitualmente mediante la transformada de distancia), podremos mostrar sus límites (segmentar) mediante la transformada divisoria. Este concepto se muestra en un ejemplo a continuación.

Sintaxis

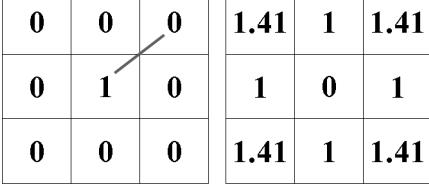
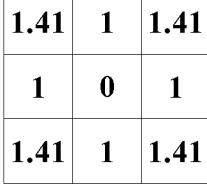
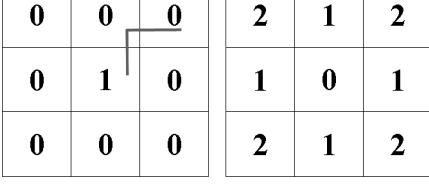
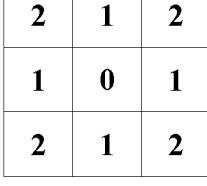
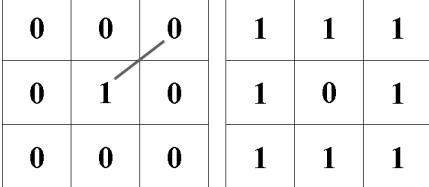
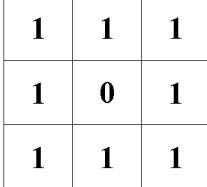
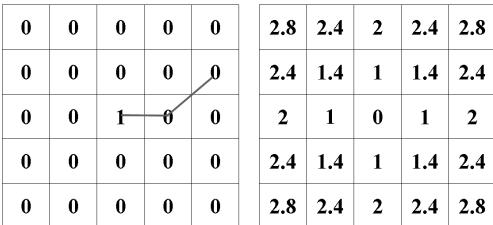
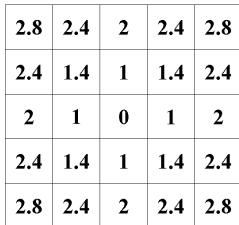
```
D = bwdist(BW)
[D,L] = bwdist(BW)
[D,L] = bwdist(BW,method)
```

Descripción

`D = bwdist(BW)` calcula la transformada de distancia de la imagen binaria `BW`. A cada pixel en la imagen `BW` se le asigna un número que es la distancia entre ese pixel y el pixel distinto de cero más cercano. `bwdist` usa por defecto la métrica euclídea. `BW` puede tener cualquier dimensión y puede ser del tipo `numeric` o `logical`, pero nunca disperso. `D` es del mismo tamaño que `BW`.

`[D, L] = bwdist(BW)` también calcula la transformada del vecino más próximo, devolviendo una matriz de etiquetas `L` cuyos elementos contienen el índice lineal del pixel distinto de cero más cercano en `BW`. `D` y `L` son matrices con el mismo tamaño que `BW`.

`[D, L] = bwdist(BW, method)` calcula la transformada de distancia especificando una métrica alternativa en la variable `method`, que se puede abbreviar y puede tomar uno de los siguientes valores:

Método	Descripción	Ilustración		
'euclidean'	Métrica: euclídea. La distancia euclídea es la distancia en línea recta entre dos píxeles. (Por defecto)			<i>Figura f.50 - Métrica euclídea</i>
'cityblock'	Métrica: <i>city block</i> . La distancia <i>city block</i> mide el camino entre los píxeles basándose en una vecindad de 4 píxeles. Los píxeles cuyos bordes se tocan distan una unidad más; los píxeles que se tocan diagonalmente distan dos unidades.			<i>Figura f.51 - Métrica city block</i>
'chessboard'	Métrica: <i>chessboard</i> . La distancia <i>chessboard</i> mide el camino basándose en una vecindad de 9 píxeles. Los píxeles cuyos bordes o esquinas se toquen distan una unidad.			<i>Figura f.52 - Métrica chessboard</i>
'quasi-euclidean'	Métrica: cuasi-euclídea. La distancia cuasi-euclídea mide la distancia euclídea total a lo largo de una serie de segmentos horizontales, verticales y diagonales.			<i>Figura f.53 - Métrica quasi-euclídea</i>

Ejemplos de uso

Ejemplo 1: Dilatación isotrópica. Mostrar los píxeles de una imagen binaria que están a menos de 25 píxeles de un píxel activo en el centro. Mostrar la imagen de la transformada de distancia de la imagen original y de la imagen final en escala de grises.

```
% Creamos una imagen binaria con un solo pixel blanco en el centro. Puede que no se vea bien el pixel y haga falta utilizar la herramienta lupa.  
BW = false(100,200);  
BW(50,100) = 1;  
imshow(BW)  
  
% Aplicamos la transformada de distancia euclídea que calcula la distancia entre cada pixel y el píxel activo más cercano, que en nuestro caso es único y está en el centro.  
  
DBW = bwdist(BW);  
  
% Seleccionamos los píxeles que distan menos de 25 unidades del píxel central (distancia <=25) y los mostramos.  
  
BW2 = DBW <= 25;  
figure, imshow(BW2)  
  
% Mostramos las transformadas de distancia en escala de grises a modo de relieve.  
  
figure, imshow(DBW,[ ])  
DBW2 = bwdist(BW2);  
figure, imshow(DBW2,[ ])
```

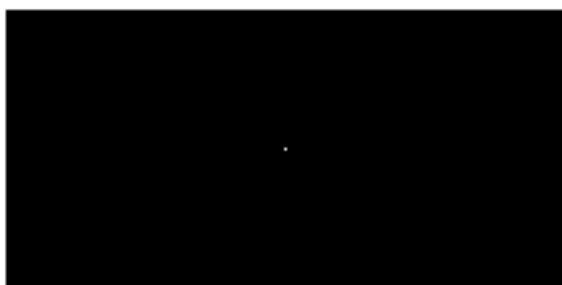


Figura f.54 - Imagen original (BW) sin procesar



Figura f.55 - Imagen procesada (BW2) con bwdist para hacer una dilatación isotrópica

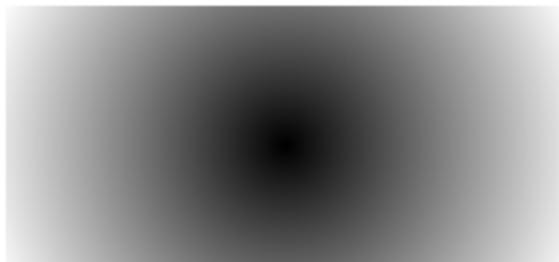


Figura f.56 - Transformada de distancia (DBW) de la imagen original BW

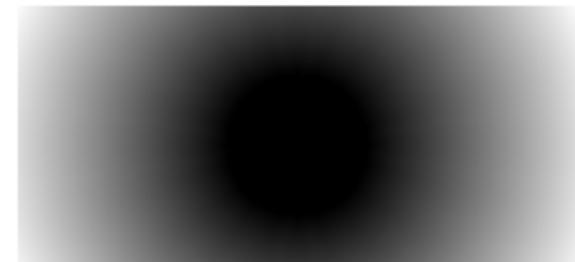


Figura f.57 - Transformada de distancia (DBW2) de la imagen procesada BW2

Ejemplo 2: Calcular la transformada del vecino más próximo.

```
bw = zeros(5,5); bw(2,2) = 1; bw(4,4) = 1
[D,L] = bwdist(bw)
bw =
    0     0     0     0     0
    0     1     0     0     0
    0     0     0     0     0
    0     0     0     1     0
    0     0     0     0     0

D =
    1.4142    1.0000    1.4142    2.2361    3.1623
    1.0000         0    1.0000    2.0000    2.2361
    1.4142    1.0000    1.4142    1.0000    1.4142
    2.2361    2.0000    1.0000         0    1.0000
    3.1623    2.2361    1.4142    1.0000    1.4142

L =
    7     7     7     7     7
    7     7     7     7    19
    7     7     7    19    19
    7     7    19    19    19
    7    19    19    19    19
```

En la matriz L de vecinos más próximos, los valores 7 y 19 representan la posición de los elementos distintos de cero utilizando indexado lineal. Por ejemplo, los píxeles que tienen valor 7 significa que su vecino distinto de cero más próximo está en la posición lineal 7.

Ejemplo 2: Segmentar los círculos de la imagen binaria utilizando la transformada de distancia en conjunto con la transformada divisoria (función watershed).

```
BW = imread('circulos.tif'); imshow(BW)

% Calculamos la transformada de distancia y la mostramos en escala de grises. Buscamos conseguir una cuenca o valle en cada objeto.

D = bwdist(BW); figure, imshow(D,[])

% Hay que conseguir cuencas (color negro) en cada objeto a segmentar y hay algunos objetos que no tienen cuenca propia. Complementamos la imagen y calculamos de nuevo la transformada de distancia.

DC = bwdist(~BW); figure, imshow(DC,[])

% Para que las áreas claras se conviertan en cuencas o valles debemos invertirlas y que sean oscuras. Negamos la transformada de distancia.

NDC = -bwdist(~BW); figure, imshow(NDC,[])

% Ahora ya tenemos una cuenca en cada objeto, así que la transformada divisoria podrá calcular sus límites. Las guarda en la matriz de etiquetas L, con las localizaciones de cada cuenca, siendo los valores igual a cero los correspondientes a las líneas divisorias. Para mostrarla conviene convertirla a color mediante la función label2rgb.
```

```
L = watershed(NDC);

% Calculamos las líneas divisorias y las mostramos.

w = L == 0; figure, imshow(w)

% Dibujamos las líneas divisorias en la imagen original para segmentar.

F = BW;
F(w) = 0; figure, imshow(F)
```

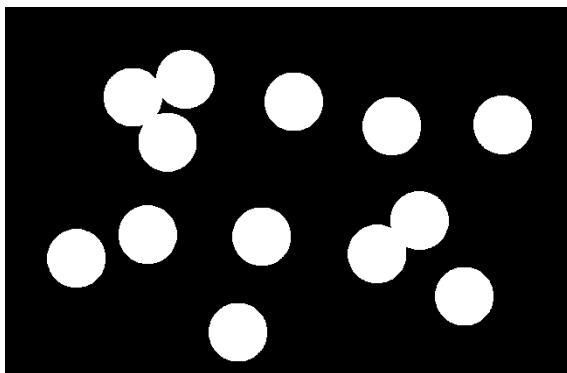


Figura f.48 - Imagen original (BW) sin procesar

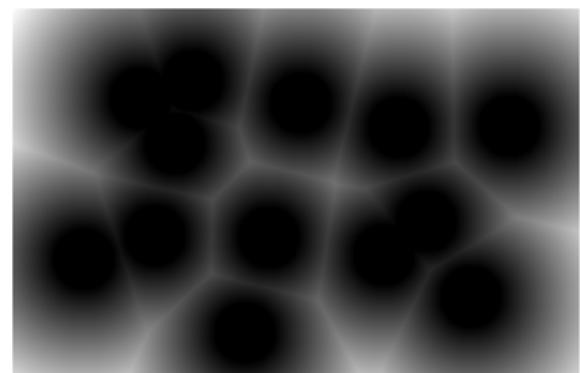


Figura f.58 - Imagen (D) de la transformada de distancia de BW

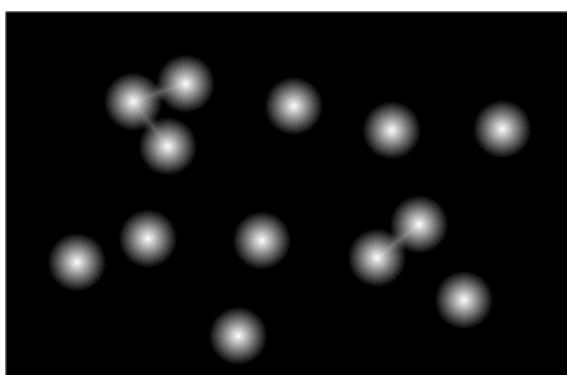


Figura f.59 - Imagen (DC) de la transformada de distancia del complemento de BW



Figura f.60 - Imagen (NDC) de la negada de la transformada de distancia del complemento de BW

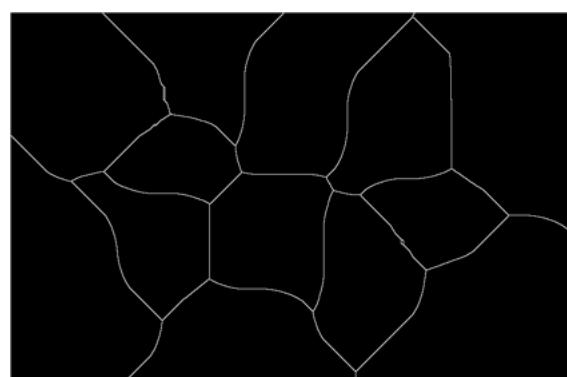


Figura f.61 - Imagen (w) de las líneas divisorias

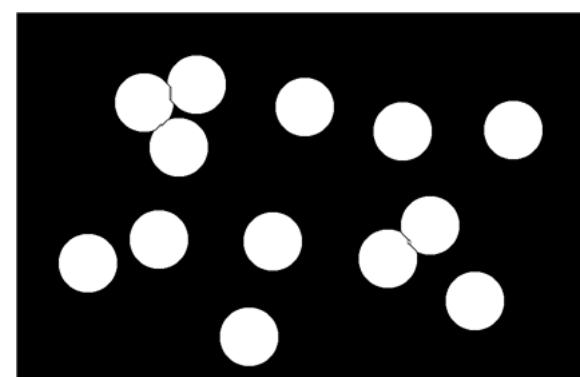


Figura f.62 - Imagen segmentada (F)

Funciones relacionadas:

`bwulterode`, `watershed`

bweuler

Número de Euler (objetos menos huecos) de una imagen binaria

Introducción

El número de Euler es una característica topográfica importante de una imagen que indica el número total de objetos en la imagen menos el número total de huecos en esos objetos.

El número de Euler permanece invariante ante las operaciones de traslación, rotación o escalado de la imagen y tiene muchas aplicaciones dentro del procesado digital de imágenes como por ejemplo el reconocimiento óptico de caracteres (OCR), el análisis de arenisca en aplicaciones geológicas, la detección de sombras o el diagnóstico médico, donde es frecuente calcular el número de Euler de imágenes de células porque el número de Euler de una célula infectada es normalmente distinto al de una sana. [8]

La rapidez del cálculo del número de Euler es por tanto esencial actualmente, de ahí que hayan aparecido recientemente muchos estudios y algoritmos que buscan hacer más eficiente su procesado en una imagen.

Sintaxis

```
eul = bweuler(BW,n)
```

Descripción

`eul = bweuler(BW,n)` devuelve el número de Euler correspondiente a la imagen binaria `BW`. El valor devuelto `eul` es un escalar cuyo valor es el número total de objetos en la imagen menos el número total de huecos en esos objetos. El argumento `n` puede tener un valor de 4 o 8, donde 4 significa conectividad-4 y 8 significa conectividad-8 (por defecto si se omite el argumento).

`BW` puede ser del tipo `numeric` o `logical` y debe ser real, no disperso y bidimensional. El valor devuelto `eul` es de tipo `double`.

Ejemplos de uso

Ejemplo 1: Calcular el número de Euler de una imagen binaria.

```
BW = imread('original.tif');
imshow(BW)
bweuler(BW)

ans =
    4
```



Figura f.48 - Imagen original (BW)

El valor de 4 corresponde con 8 objetos en blanco y 4 huecos dentro de los objetos.

Para contar solo el número de objetos omitiendo los huecos se puede utilizar la función bwlabel con la sintaxis: [etiquetas, numobj] = bwlabel(I,8);

Ejemplo 2: Calcular el número de Euler de la imagen anterior utilizando la función regionprops. Mostrar en una imagen solo los objetos con huecos y en otra solo los objetos sin huecos.

```
BW = imread('original.tif');
L = bwlabel(BW,8);

e = regionprops(L,'EulerNumber');
euler=[e.EulerNumber];
totale = sum(euler)

imshow(ismember(L, find(euler < 1)))
figure, imshow(ismember(L, find(euler > 0)))
```



Figura f.63 - Imagen (w) de las líneas divisorias

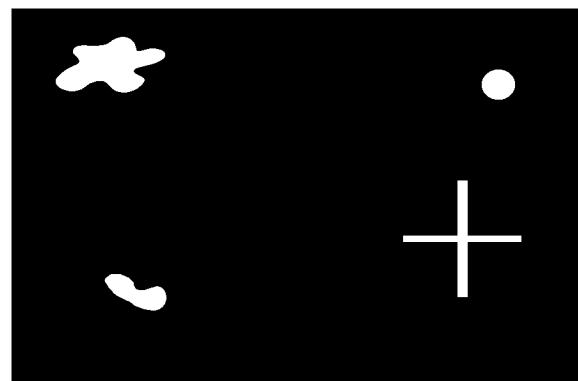


Figura f.64 - Imagen segmentada (F)

Ejemplo 3: Calcular el número de Euler de las siguientes imágenes binarias que representan la red trabecular de dos pacientes, el primero un paciente sano y el segundo un paciente con osteoporosis. Las cavidades encerradas y la red trabecular es más abundante en el paciente sano, lo que conlleva un menor número de Euler.

```
BW1 = imread('sano.tif'); imshow(BW1)
BW2 = imread('osteо.tif'); figure, imshow(BW2)
eulersano = bweuler(BW1)
eulerosteо = bweuler(BW2)

eulersano =
    7

eulerosteо =
    15
```

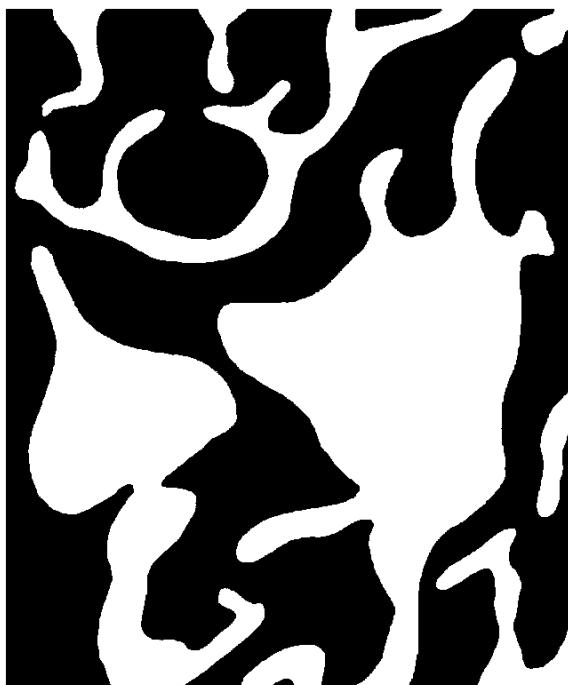


Figura f.65 - Imagen (BW1) que representa a paciente sano [9]

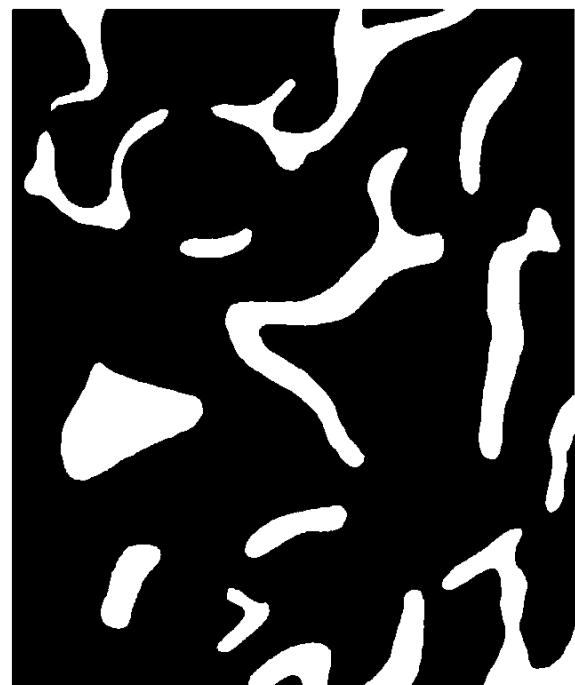


Figura f.66 - Imagen (BW2) que representa a paciente con osteoporosis [9]

Funciones relacionadas

bwmorph, bwperim

bwhitmiss

Transformada de localización - búsqueda y omisión de píxeles usando patrones de vecindad determinados en una imagen binaria

Introducción

La función `bwhitmiss` se engloba dentro de las funciones utilizadas en el procesado morfológico de una imagen, especialmente en las técnicas de dilatación y erosión. Es útil cuando se necesita la identificación de una determinada configuración de píxeles, como píxeles aislados de la imagen o píxeles que son los puntos finales de segmentos. Cuando los elementos estructurantes o patrones utilizados son pequeños, resulta más rápido el uso de tablas de consulta mediante las funciones `makelut` y `applylut`.

En morfología matemática, la transformación de localización es una operación que detecta una determinada configuración (o patrón) en un imagen binaria, con el operador morfológico erosión y un par de elementos estructurantes disjuntos. El resultado de la transformación de localización es el conjunto de posiciones donde el primer elemento estructurante cabe en el primer plano de la imagen de entrada y el segundo elemento de estructuración cabe en el segundo plano o fondo de la imagen.

Sintaxis

```
BW2 = bwhitmiss(BW1,SE1,SE2)
BW2 = bwhitmiss(BW1,INTERVAL)
```

Descripción

`BW2 = bwhitmiss(BW1,SE1,SE2)` realiza la transformada de localización definida por las estructuras `SE1` y `SE2`. La operación mantiene igual los píxeles cuyos vecinos cumplan con el patrón o figura definida en `SE1` y que a su vez no cumplan el definido en `SE2`. `SE1` y `SE2` pueden ser objetos estructurales planos creados por la función `strel` o arrays de vecindades extraídos del objeto estructurante mediante la función `getnhood`. Las vecindades de `SE1` y `SE2` no deben tener ningún elemento solapado. La sintaxis `bwhitmiss(BW1,SE1,SE2)` es equivalente a `imerode(BW1,SE1) & imerode(~BW1,SE2)`.

`BW2 = bwhitmiss(BW1,INTERVAL)` realiza la operación de ‘acierto y fallo’ definida a partir de un array `INTERVAL` que se pasa como parámetro y cuyos elementos son 1, 0 o -1.

Los valores igual a 1 forman SE1, los valores igual a -1 forman SE2 y los valores igual a 0 se ignoran. La sintaxis bwhitmiss(BW1, INTERVAL) es equivalente a bwhitmiss(BW1, INTERVAL == 1, INTERVAL == -1).

BW1 puede ser un array del tipo numeric o logical de cualquier dimensión y debe ser no disperso. BW2 siempre es un array logical del mismo tamaño que BW1. SE1 y SE2 deben ser objetos planos STREL o arrays tipo numeric o logical con solamente unos y ceros. INTERVAL debe ser un array con 1's, 0's, y -1's.

Ejemplos de uso

Ejemplo 1: Encontrar el patrón P en BW utilizando la función bwhitmiss y después utilizando tablas de consulta mediante la función applylut.

```
P = [0 1 0
      1 1 1
      0 1 0];
BW = [0 1 0 0 0 0 0
      1 1 1 0 0 1 0
      0 1 0 0 0 0 0
      0 0 0 0 0 0 0
      1 1 1 0 0 1 0
      1 1 1 0 1 1 1
      1 1 1 0 0 0 0];
% Hacemos el cálculo primero con bwhitmiss.
% Lo primero es crear los patrones binarios que queremos encontrar en BW.
% Primero el que queremos que se cumpla y luego el que queremos evitar:
B1 = [0 1 0
      1 1 1
      0 1 0];
B2 = [1 0 1
      0 0 0
      1 0 1];
S1 = bwhitmiss(BW,B1,B2)
% Hacemos ahora el cálculo con tablas de consulta.
% calculamos el valor de índice del patrón a buscar: 0x1 + 1x2 + 0+4 +
1x8 + 1x16 + 1x32 + 0x64 + 1x128 + 0x256 = 2 + 8 + 16 + 32 + 128 = 186.
% Generamos la tabla de consulta y la aplicamos sobre la imagen.
lut = zeros(512,1);
lut(187) = 1;
S2 = applylut(BW,lut)
```

```
BW =
    0     1     0     0     0     0     0
    1     1     1     0     0     1     0
    0     1     0     0     0     0     0
    0     0     0     0     0     0     0
    1     1     1     0     0     1     0
    1     1     1     0     1     1     1
    1     1     1     0     0     0     0

S1 =
    0     0     0     0     0     0     0
    0     1     0     0     0     0     0
    0     0     0     0     0     0     0
    0     0     0     0     0     0     0
    0     0     0     0     0     0     0
    0     0     0     0     0     0     0
    0     0     0     0     0     0     0

S2 =
    0     0     0     0     0     0     0
    0     1     0     0     0     0     0
    0     0     0     0     0     0     0
    0     0     0     0     0     0     0
    0     0     0     0     0     0     0
    0     0     0     0     0     0     0
    0     0     0     0     0     0     0
```

Ejemplo 2: Activar el cero central de una vecindad de 4 píxeles (un píxel cero con píxeles vecinos activos al norte, sur, este y oeste) usando `bwhitmiss` para encontrar los píxeles que debemos activar en un array BW.

```
BW = [1 1 0 0 0 0 0
      1 0 1 0 0 1 0
      0 1 0 0 0 0 0
      0 0 0 0 0 0 0
      1 1 1 0 0 1 1
      1 0 1 0 1 0 1
      1 1 1 0 1 0 1]
```

% Lo primero es crear el patrón que queremos encontrar en BW. Para buscar una vecindad de 3-por-3 siendo activos los píxeles al norte, sur, este y oeste, debemos usar la siguiente estructura:

```
e1 = [0 1 0
      1 0 1
      0 1 0];
```

% Ahora definimos la estructura que representa la configuración a evitar o excluir. Para que el pixel central de una vecindad de 3-por-3 sea cero, usaremos esta estructura:

```
e2 = [0 0 0
      0 1 0
      0 0 0];
```

% Aplicamos `bwhitmiss` a la imagen binaria para que cree un array con los píxeles que cumplen estos patrones de vecinos.

```
encontpixel = bwhitmiss(BW,e1,e2)

% Ahora, si queremos que esos píxeles encontrados cambien de valor de
% cero a uno en la imagen original, no tenemos más que crear un array con
% la operación OR exclusiva del array original y del de píxeles
% encontrados:

BW2 = BW, encontpixel

BW =
    1     1     0     0     0     0     0
    1     0     1     0     0     1     0
    0     1     0     0     0     0     0
    0     0     0     0     0     0     0
    1     1     1     0     0     1     1
    1     0     1     0     1     0     1
    1     1     1     0     1     0     1

encontpixel =
    0     0     0     0     0     0     0
    0     1     0     0     0     0     0
    0     0     0     0     0     0     0
    0     0     0     0     0     0     0
    0     0     0     0     0     0     0
    0     1     0     0     0     0     0
    0     0     0     0     0     0     0

BW2 =
    1     1     0     0     0     0     0
    1     1     1     0     0     1     0
    0     1     0     0     0     0     0
    0     0     0     0     0     0     0
    1     1     1     0     0     1     1
    1     1     1     0     1     0     1
    1     1     1     0     1     0     1
```

Ejemplo 2: Encontrar los rectángulos con tamaño 9x6 píxeles en la imagen binaria.

```
I=imread('cuadros.tif');
imshow(I)

% Creamos estructuras a buscar y evitar.

B=zeros(11,9);
B(2:10, 2:7)=1;
B1=B

B=ones(11,9);
B(2:10,2:7)=0; B(1:11,9)=0;
B2=B

% Aplicamos bwhitmiss y buscamos el rectángulo que hay en las posiciones
% encontradas.

C=bwhitmiss(I,B1,B2);
[fila col]=find(C);
C(fila-4:fila+4,col-3:col+2)=1;
figure, imshow(C)
```

```

B1 =
0   0   0   0   0   0   0   0
0   1   1   1   1   1   1   0
0   1   1   1   1   1   1   0
0   1   1   1   1   1   1   0
0   1   1   1   1   1   1   0
0   1   1   1   1   1   1   0
0   1   1   1   1   1   1   0
0   1   1   1   1   1   1   0
0   1   1   1   1   1   1   0
0   1   1   1   1   1   1   0
0   1   1   1   1   1   1   0
0   1   1   1   1   1   1   0
0   0   0   0   0   0   0   0

B2 =
1   1   1   1   1   1   1   0
1   0   0   0   0   0   0   1
1   0   0   0   0   0   0   0
1   0   0   0   0   0   0   1
1   0   0   0   0   0   0   0
1   0   0   0   0   0   0   1
1   0   0   0   0   0   0   0
1   0   0   0   0   0   0   1
1   0   0   0   0   0   0   0
1   0   0   0   0   0   0   1
1   0   0   0   0   0   0   0
1   1   1   1   1   1   1   0

```

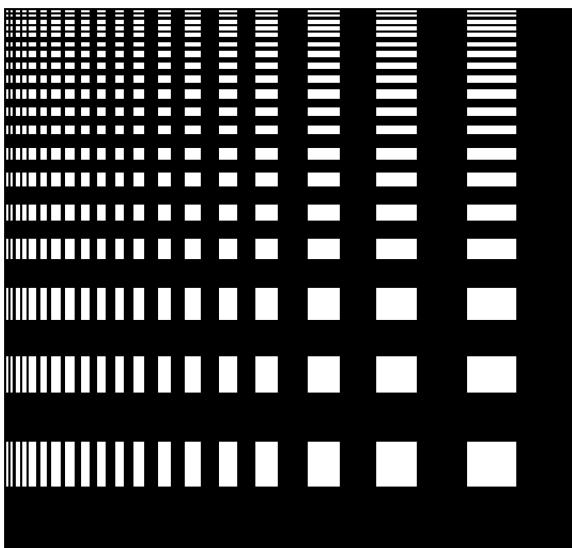


Figura f.67 - Imagen original (I)

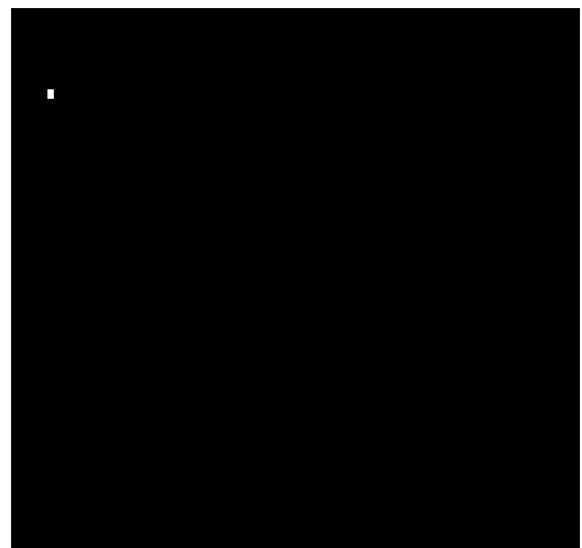


Figura f.68 - Imagen (C)

Funciones relacionadas

`applylut`, `imdilate`, `imerode`, `makelut`, `strel`

bwlabel

Etiquetado de componentes conectadas en una imagen binaria en 2-D

Introducción

Las funciones `bwlabel`, `bwlabeln` y `bwconncomp` calculan las componentes conectadas en imágenes binarias. `bwconncomp` reemplaza el uso de `bwlabel` y `bwlabeln` usando mucha menos memoria y es normalmente más rápida que las otras funciones. Esta es una comparativa de las funciones:

Función	Dimensión de entrada	Tipo de salida	Uso de memoria	Conectividad
<code>bwlabel</code>	2-D	Matriz de etiquetas con precisión doble.	Alto	4 o 8
<code>bwlabeln</code>	N-D	Matriz de etiquetas con precisión doble.	Alto	Cualquiera
<code>bwconncomp</code>	N-D	Estructura CC.	Bajo	Cualquiera

Sintaxis

```
L = bwlabel(BW, n)  
[L, num] = bwlabel(BW, n)
```

Descripción

`L = bwlabel(BW, n)` devuelve una matriz `L` denominada matriz de etiquetas, del mismo tamaño que `BW` y con etiquetas para todos los elementos conectados de la imagen `BW`. El argumento `n` puede tener un valor de 4 o 8, donde 4 significa conectividad-4 y 8 significa conectividad-8 (por defecto si se omite el argumento).

`BW` puede ser del tipo `numeric` o `logical` y debe ser real, bidimensional y no disperso. `L` es de tipo `double`.

Los elementos de `L` son valores enteros, más grandes o iguales que 0. Los píxeles etiquetados con 0 son el fondo de la imagen. Los píxeles etiquetados con 1 crean un objeto. Los píxeles etiquetados con un 2 forman un segundo objeto, y así consecutivamente.

[L, num] = bwlabel(BW, n) devuelve a la variable num el número de elementos conectados encontrados en BW.

Para medir las características de objetos dentro de una imagen binaria utilizando la función regionprops con conectividad por defecto, usar BW directamente en regionprops (p. ej. regionprops(BW)).

Utilizar la función find junto a bwlabel para obtener vectores de índices de fila y columna para los píxeles de un objeto específico. Por ejemplo, para obtener las coordenadas para los píxeles del objeto 2 se debe escribir:

```
[r, c] = find(bwlabel(BW)==2)
```

Se puede mostrar la matriz de salida como una imagen indexada con la función label2rgb. Cada objeto aparecerá con un color diferente y serán más fáciles de reconocer.

Ejemplos de uso

Ejemplo 1: Etiquetar objetos de un array usando conectividad de 4 y 8 píxeles. Notar que los objetos 2 y 3 se cuentan como un único objeto usando conectividad-8 en vez de contarse como dos objetos por separado como sucede con conectividad-4.

```
BW = logical ([1 1 1 0 0 0 0 0
                1 1 1 0 1 1 0 0
                1 1 1 0 1 1 0 0
                1 1 1 0 0 0 1 0
                1 1 1 0 0 0 1 0
                1 1 1 0 0 0 1 0
                1 1 1 0 0 0 1 0
                1 1 1 0 0 1 1 0
                1 1 1 0 0 0 0 0]);
% Conectividad 4.
L4 = bwlabel(BW,4)
% Buscamos los índices de fila y columna para todos los píxeles del
% segundo objeto.
[r4, c4] = find(L4 == 2);
r4c4 = [r4 c4]
% Conectividad 8.
L8 = bwlabel(BW,8)
[r8, c8] = find(L8 == 2);
r8c8 = [r8 c8]
```

```
L4 =  
1 1 1 0 0 0 0 0  
1 1 1 0 2 2 0 0  
1 1 1 0 2 2 0 0  
1 1 1 0 0 0 3 0  
1 1 1 0 0 0 3 0  
1 1 1 0 0 0 3 0  
1 1 1 0 0 0 3 0  
1 1 1 0 0 0 3 0  
1 1 1 0 0 0 0 0  
  
r4c4 =  
2 5  
3 5  
2 6  
3 6  
  
L8 =  
1 1 1 0 0 0 0 0  
1 1 1 0 2 2 0 0  
1 1 1 0 2 2 0 0  
1 1 1 0 0 0 2 0  
1 1 1 0 0 0 2 0  
1 1 1 0 0 0 2 0  
1 1 1 0 0 2 2 0  
1 1 1 0 0 0 0 0  
  
r8c8 =  
2 5  
3 5  
2 6  
3 6  
7 6  
4 7  
5 7  
6 7  
7 7
```

Funciones relacionadas

[bwconncomp](#), [bwlabeln](#), [bwselect](#), [labelmatrix](#), [label2rgb](#), [regionprops](#)

bwlabeln

Etiquetado de componentes conectadas usando conectividad-n en una imagen binaria

Introducción

Las funciones `bwlabel`, `bwlabeln` y `bwconncomp` calculan las componentes conectadas en imágenes binarias. `bwconncomp` reemplaza el uso de `bwlabel` y `bwlabeln` usando mucha menos memoria y es normalmente más rápida que las otras funciones. Esta es una comparativa de las funciones:

Función	Dimensión de entrada	Tipo de salida	Uso de memoria	Conectividad
<code>bwlabel</code>	2-D	Matriz de etiquetas con precisión doble.	Alto	4 o 8
<code>bwlabeln</code>	N-D	Matriz de etiquetas con precisión doble.	Alto	Cualquiera
<code>bwconncomp</code>	N-D	Estructura CC.	Bajo	Cualquiera

Sintaxis

```
L = bwlabeln(BW)
[L, NUM] = bwlabeln(BW)
[L, NUM] = bwlabeln(BW, conn)
```

Descripción

`L = bwlabeln(BW)` devuelve una matriz de etiquetas `L` conteniendo etiquetas para todos los elementos conectados de la imagen `BW`. La imagen de entrada puede tener cualquier dimensión. `L` tiene el mismo tamaño que `BW`. Los elementos de `L` son enteros mayores o iguales que 0. Los píxeles etiquetados con 0 son el fondo de la imagen. Los píxeles etiquetados con 1 crean un objeto. Los píxeles etiquetados con un 2 forman un segundo objeto, y así consecutivamente. La conectividad por defecto es 8 para dos dimensiones, 26 para tres dimensiones y `conndef(ndims(BW), 'maximal')` para dimensiones mayores.

`BW` puede ser del tipo `numeric` o `logical` y debe ser real y no disperso. `L` es de tipo `double`.

[L, NUM] = bwlabeln(BW) devuelve en NUM el número de objetos conectados encontrados en BW.

[L, NUM] = bwlabeln(BW, conn) especifica la conectividad deseada. conn puede tener cualquier uno de los siguientes valores escalares (siempre simétrica respecto su elemento central):

Valor	Significado
Conejividades Bidimensionales	
4	Vecindad de 4 píxeles: Un par de píxeles colindantes forman parte del mismo objeto solo si están activos y conectados en vertical u horizontal.
8	Vecindad de 8 píxeles: Un par de píxeles colindantes forman parte del mismo objeto solo si están activos. (Por defecto)
Conejividades Tridimensionales	
6	Vecindad de 6 píxeles.
18	Vecindad de 18 píxeles.
26	Vecindad de 26 píxeles. (Por defecto)
<p>Nota: La conectividad se puede definir para cualquier dimensión de forma más general utilizando para conn una matriz de 3x3x...x3 de ceros y unos. Los unos definen las localizaciones de los píxeles vecinos relativos al elemento central de conn. Por ejemplo, conn=ones(3) define una conectividad de 8 píxeles.</p>	

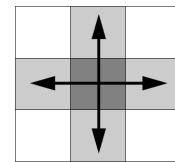


Figura f.32 - vecindad 4

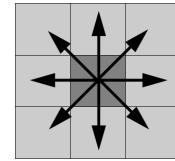


Figura f.33 - vecindad 8

Utilizar la función `find` junto a `bwlabeln` para obtener vectores de índices de fila y columna para los píxeles de un objeto específico. Por ejemplo, para obtener las coordenadas para los píxeles del objeto 2 se debe escribir:

```
[r, c] = find(bwlabeln(BW)==2)
```

Se puede mostrar la matriz de salida como una imagen indexada con la función `label2rgb`. Cada objeto aparecerá con un color diferente y serán más fáciles de reconocer.

Para medir las características de objetos dentro de una imagen binaria utilizando la función `regionprops` con conectividad por defecto, usar `BW` directamente en `regionprops` (p. ej. `regionprops(BW)`).

Funciones relacionadas

`bwconncomp`, `bwlabel`, `labelmatrix`, `label2rgb`, `regionprops`

bwmorph

Operaciones morfológicas en imágenes binarias

Introducción

La función `bwmorph` implementa una serie de operaciones morfológicas muy útiles basadas en combinaciones de dilataciones, erosiones y operaciones de búsqueda en tablas.

Sintaxis

```
BW2 = bwmorph(BW,operation)
BW2 = bwmorph(BW,operation,n)
```

Descripción

`BW2 = bwmorph(BW,operation)` aplica una operación morfológica a la imagen binaria `BW`.

`BW2 = bwmorph(BW,operation,n)` aplica la operación `n` veces. `n` puede ser `Inf` (infinito), en cuyo caso la operación se repite hasta que la imagen no cambie más (repetición hasta estabilidad). Esencial cuando la operación que realizamos debe repetirse varias veces pero desconocemos el número de veces a repetir.

La imagen de entrada `BW` puede ser del tipo `numeric` o `logical`. Debe ser 2-D, real y no dispersa. La imagen de salida `BW2` es del tipo `logical`. `Operation` es una cadena que puede tener uno de los siguientes valores:

Operación	Descripción
'bothat'	Realiza la operación morfológica <i>black top-hat</i> o <i>bottom-hat</i> (sombbrero de copa negro), dual a <i>white top-hat</i> o <i>top-hat</i> (sombbrero de copa blanco), devolviendo la diferencia entre la clausura de una imagen por un elemento estructurante (la colección de partes del fondo de la imagen que corresponden con un elemento estructurante o patrón particular) y la imagen original. Es una dilatación seguida de una erosión. Se utiliza un elemento estructurante de 3-por-3. Para utilizar otro elemento estructurante distinto o aplicarlo sobre una imagen en escala de grises se debe utilizar la función <code>imbothat</code> . La operación devuelve una imagen con los objetos que son menores que el elemento estructurante y que son más oscuros que sus vecinos (se debe aumentar el tamaño del elemento estructurante para extraer

	objetos mayores). Es por tanto útil para extraer pequeños objetos o detalles oscuros sobre fondo claro y también para corregir la iluminación desigual cuando el fondo es claro. Además, se puede mejorar el contraste de una imagen sumando la imagen original y la imagen filtrada con <i>top-hat</i> y posteriormente restándole la imagen filtrada con <i>bottom-hat</i> .														
'branchpoints'	Encuentra los puntos de intersección de un esqueleto. Por ejemplo: <table style="margin-left: 20px; margin-right: 20px;"> <tr><td>0 0 1 0 0</td><td>0 0 0 0 0</td></tr> <tr><td>0 0 1 0 0</td><td>se convierte en</td><td>0 0 0 0 0</td></tr> <tr><td>1 1 1 1 1</td><td></td><td>0 0 1 0 0</td></tr> <tr><td>0 0 1 0 0</td><td></td><td>0 0 0 0 0</td></tr> <tr><td>0 0 1 0 0</td><td></td><td>0 0 0 0 0</td></tr> </table>	0 0 1 0 0	0 0 0 0 0	0 0 1 0 0	se convierte en	0 0 0 0 0	1 1 1 1 1		0 0 1 0 0	0 0 1 0 0		0 0 0 0 0	0 0 1 0 0		0 0 0 0 0
0 0 1 0 0	0 0 0 0 0														
0 0 1 0 0	se convierte en	0 0 0 0 0													
1 1 1 1 1		0 0 1 0 0													
0 0 1 0 0		0 0 0 0 0													
0 0 1 0 0		0 0 0 0 0													
'bridge'	Conecta píxeles no conectados (pone a 1 los píxeles de valor 0 que tengan dos vecinos distintos de cero no conectados). Por ejemplo: <table style="margin-left: 20px; margin-right: 20px;"> <tr><td>1 0 0</td><td>1 1 0</td></tr> <tr><td>1 0 1</td><td>se convierte en</td><td>1 1 1</td></tr> <tr><td>0 0 1</td><td></td><td>0 1 1</td></tr> </table>	1 0 0	1 1 0	1 0 1	se convierte en	1 1 1	0 0 1		0 1 1						
1 0 0	1 1 0														
1 0 1	se convierte en	1 1 1													
0 0 1		0 1 1													
'clean'	Elimina los píxeles aislados (píxeles individuales con valor 1 que están rodeados por ceros). Por ejemplo, el pixel central de esta estructura: <table style="margin-left: 20px; margin-right: 20px;"> <tr><td>0 0 0</td></tr> <tr><td>0 1 0</td></tr> <tr><td>0 0 0</td></tr> </table>	0 0 0	0 1 0	0 0 0											
0 0 0															
0 1 0															
0 0 0															
'close'	Realiza una clausura morfológica (dilatación seguida de una erosión). Elimina pequeños huecos (rellenándolos) y une componentes conexas cercanas. Se utiliza un elemento estructurante de 3-por-3. Para utilizar otro elemento estructurante distinto o aplicarlo sobre una imagen en escala de grises se debe utilizar la función <code>imclose</code> .														
'diag'	Utiliza relleno diagonal para eliminar la conectividad-8 del fondo. Por ejemplo: <table style="margin-left: 20px; margin-right: 20px;"> <tr><td>0 1 0</td><td>0 1 0</td></tr> <tr><td>1 0 0</td><td>se convierte en</td><td>1 1 0</td></tr> <tr><td>0 0 0</td><td></td><td>0 0 0</td></tr> </table>	0 1 0	0 1 0	1 0 0	se convierte en	1 1 0	0 0 0		0 0 0						
0 1 0	0 1 0														
1 0 0	se convierte en	1 1 0													
0 0 0		0 0 0													
'dilate'	Realiza una dilatación utilizando el elemento estructural <code>ones(3)</code> . Para utilizar otro elemento estructurante distinto o aplicarlo sobre una imagen en escala de grises se debe utilizar la función <code>imdilate</code> .														
'endpoints'	Encuentra los píxeles límite de un esqueleto. Por ejemplo: <table style="margin-left: 20px; margin-right: 20px;"> <tr><td>1 0 0 0</td><td>1 0 0 0</td></tr> <tr><td>0 1 0 0</td><td>se convierte en</td><td>0 0 0 0</td></tr> <tr><td>0 0 1 0</td><td></td><td>0 0 1 0</td></tr> <tr><td>0 0 0 0</td><td></td><td>0 0 0 0</td></tr> </table>	1 0 0 0	1 0 0 0	0 1 0 0	se convierte en	0 0 0 0	0 0 1 0		0 0 1 0	0 0 0 0		0 0 0 0			
1 0 0 0	1 0 0 0														
0 1 0 0	se convierte en	0 0 0 0													
0 0 1 0		0 0 1 0													
0 0 0 0		0 0 0 0													
'erode'	Realiza una erosión utilizando el elemento estructural <code>ones(3)</code> . Para utilizar otro elemento estructurante distinto o aplicarlo sobre una imagen en escala de grises se debe utilizar la función <code>imerode</code> .														
'fill'	Rellena píxeles interiores aislados (0s individuales que están rodeados de 1s). Por ejemplo, el pixel central de esta estructura:														

	<pre> 1 1 1 1 0 1 1 1 1 </pre> <p>Para llenar huecos más grandes o aplicarlo sobre una imagen en escala de grises se debe utilizar la función <code>imfill</code>.</p>
'hbreak'	<p>Elimina píxeles conectados con conectividad-H. Por ejemplo:</p> <pre> 1 1 1 1 1 1 0 1 0 se convierte en 0 0 0 1 1 1 1 1 1 </pre>
'majority'	<p>Pone un pixel a 1 si cinco o más píxeles en su vecindad de 3-por-3 son 1s; en caso contrario pone el pixel a 0. Por ejemplo:</p> <pre> 1 1 1 0 0 0 0 1 0 se convierte en 1 1 1 1 1 1 0 0 0 </pre>
'open'	<p>Realiza una apertura morfológica (erosión seguida de dilatación). La apertura generalmente suaviza los contornos de una imagen y elimina pequeños salientes. También puede eliminar franjas o zonas de un objeto que sean “más estrechas” que el elemento estructural.</p> <p>Se utiliza un elemento estructurante de 3-por-3. Para utilizar otro elemento estructurante distinto o aplicarlo sobre una imagen en escala de grises se debe utilizar la función <code>imopen</code>.</p>
'remove'	<p>Elimina píxeles interiores. Pone un pixel a 0 si todos sus vecinos con conectividad-4 están a 1, dejando solo los píxeles frontera activos. Por ejemplo:</p> <pre> 1 1 1 1 1 1 1 1 1 se convierte en 1 0 1 1 1 1 1 1 1 </pre>
'shrink'	<p>Con <code>n = Inf</code> (infinito) contrae objetos a puntos. Elimina los píxeles hasta que los objetos sin huecos se reduzcan a un punto y los objetos con huecos se reduzcan a un anillo entre cada hueco y la frontera exterior. Esta opción mantiene el número de Euler. Por ejemplo:</p> <pre> 1 1 1 0 0 0 1 1 1 se convierte en 0 1 0 1 1 1 0 0 0 1 1 1 0 1 0 1 0 1 se convierte en 1 0 1 1 1 1 0 1 0 </pre>
'skel'	<p>Forma el esqueleto de una imagen (esqueletización). Junto con la operación 'thin' es una de las operaciones más habituales para reducir objetos binarios a un conjunto de líneas que aportan información de relevancia sobre las formas originales. Es como una erosión que finaliza cuando se llega a la línea final de los píxeles del objeto.</p> <p>Con <code>n = Inf</code> (infinito) elimina los píxeles en la frontera de los objetos pero no permite romper los objetos. Los píxeles restantes forman el esqueleto de la imagen. Esta opción mantiene el número de Euler. Por ejemplo:</p>

	<pre> 1 1 1 1 1 1 se convierte en 1 0 0 1 1 1 1 1 1 1 0 0 </pre>
'spur'	Pone un pixel a 0 si solo tiene un pixel con conectividad-8 en su vecindad. Por ejemplo: <pre> 0 0 0 0 0 0 0 0 0 0 1 0 se convierte en 0 0 0 0 0 1 0 0 0 1 0 0 1 1 0 0 1 1 0 0 </pre>
'thicken'	Con $n = \text{Inf}$ (infinito) engorda objetos añadiendo píxeles al exterior del objeto hasta que los píxeles que antes no estaban conectados con conectividad-8 ahora lo estén. Esta opción mantiene el número de Euler. Por ejemplo: <pre> 0 0 0 0 1 0 se convierte en 1 1 1 0 0 0 1 1 1 </pre>
'thin'	Es una de las operaciones más habituales para reducir objetos binarios a un conjunto de líneas que aportan información de relevancia sobre las formas originales. Con $n = \text{Inf}$ (infinito) encoge objetos hasta formar líneas de un solo pixel de ancho. Elimina píxeles hasta que un objeto sin huecos se reduzca a un trazo mínimo y hasta que un objeto con huecos se reduzca a un anillo entre cada hueco y la frontera exterior. Se mantiene el número de Euler. Por ejemplo: <pre> 1 1 1 1 1 1 se convierte en 0 0 0 1 1 1 0 1 0 0 0 0 1 1 1 1 0 1 se convierte en 0 1 0 1 1 1 1 0 1 0 1 0 </pre>
'tophat'	Realiza la operación morfológica <i>white top-hat</i> (sombbrero de copa blanco), contraria a <i>black top-hat o bottom-hat</i> (sombbrero de copa negro) devolviendo la diferencia entre la imagen original y su apertura por un elemento estructurante (la colección de partes del frente de una imagen que corresponden con un elemento estructurante o patrón particular). Es una erosión seguida de una dilatación. Se utiliza un elemento estructurante de 3-por-3. Para utilizar otro elemento estructurante distinto o aplicarlo sobre una imagen en escala de grises se debe utilizar la función <code>imtophat</code> . La operación devuelve una imagen con los objetos que son menores que el elemento estructurante y que son más brillantes que sus vecinos (aumentar el tamaño del elemento estructurante para extraer objetos mayores). Es por tanto útil para extraer pequeños objetos o detalles claros sobre fondo oscuro y también para corregir la iluminación desigual cuando el fondo es oscuro. Además, se puede mejorar el contraste sumando la imagen original y la imagen filtrada con <i>top-hat</i>

y posteriormente restándole la imagen filtrada con *bottom-hat*.

Ejemplos de uso

Ejemplo 1: Aplicar la operación '*thin*' infinita para encoger una imagen binaria de una huella digital hasta conseguir las líneas más estrechas posibles.

```
BW = imread('finger.tif');
BW2 = bwmorph(BW, 'thin', Inf);

imshow(BW)
figure, imshow(BW2)
```

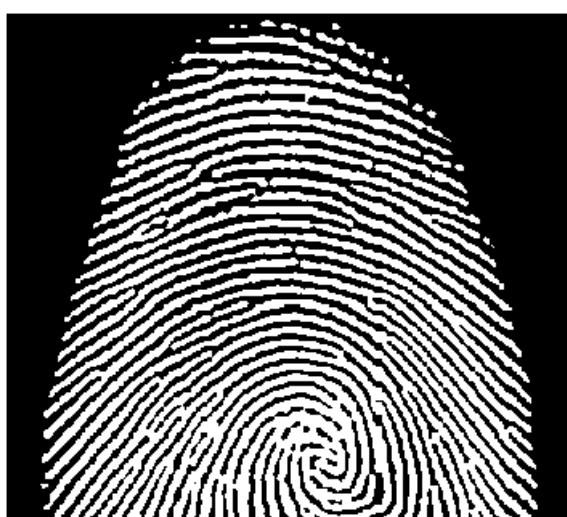


Figura f.69 - Imagen original (BW)

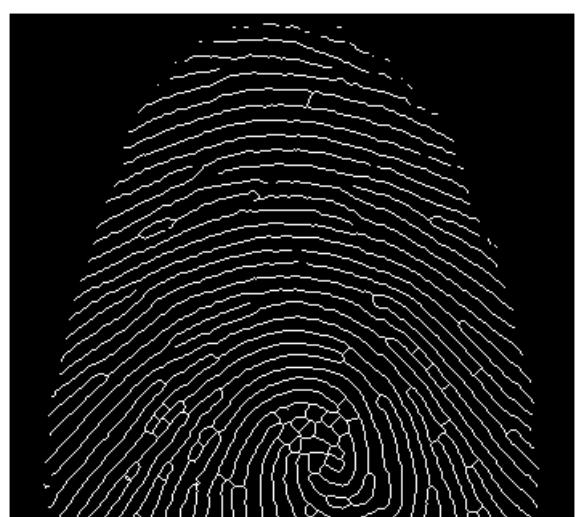


Figura f.70 - Imagen procesada (BW2)
con bwmorph parámetro 'thin'

Ejemplo 2: Aplicar la operación '*remove*' de la función *bwmorph* a la imagen.

```
BW = imread('circulos.tif');
BW2 = bwmorph(BW, 'remove');

imshow(BW)
figure, imshow(BW2)
```

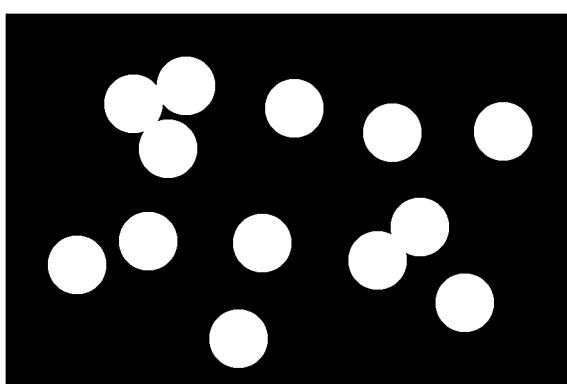


Figura f.48 - Imagen original (BW)

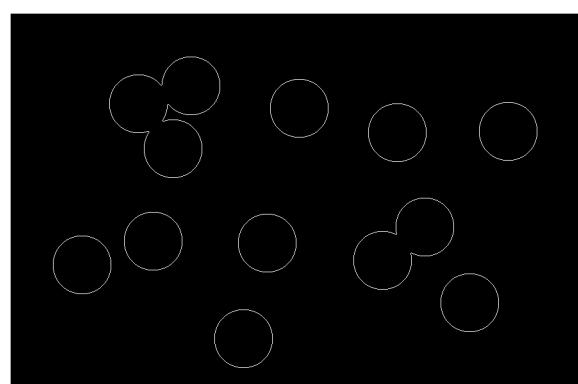


Figura f.71 - Imagen procesada (BW2)
con bwmorph parámetro 'remove'

Ejemplo 3: Aplicar la operación de esqueletizado ('skel') infinito a la imagen, aplicarle al resultado la operación de engorde ('thicken') y finalmente superponer el esqueleto en la imagen original.

```
BW = imread('formas.tif');
BW2 = bwmorph(BW, 'skel', Inf);
BW3 = bwmorph(BW2, 'thicken' );
BW4 = BW - BW3;

figure, imshow(BW)
figure, imshow(BW2)
figure, imshow(BW3)
figure, imshow(BW4)
```



Figura f.39 - Imagen original (BW)

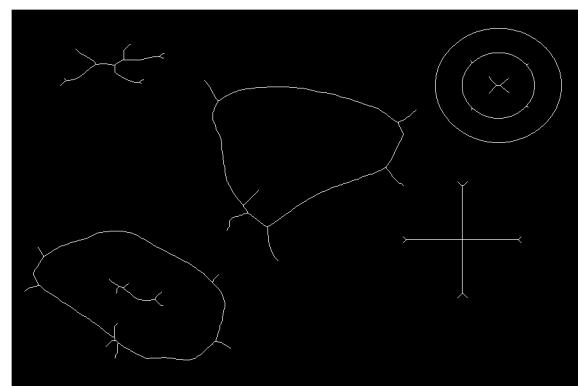


Figura f.72 - Imagen procesada (BW2) con bwmorph parámetro 'skel' infinito

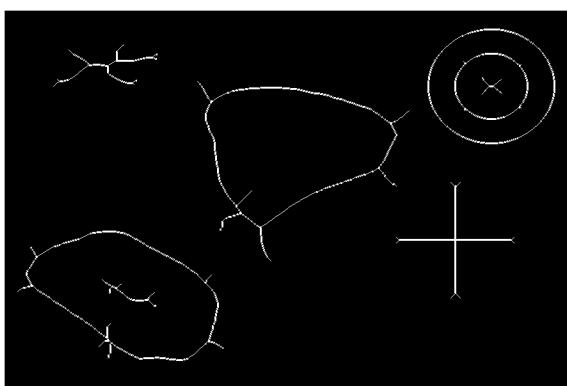


Figura f.73 - Imagen procesada (BW3) con bwmorph parámetro 'thicken'

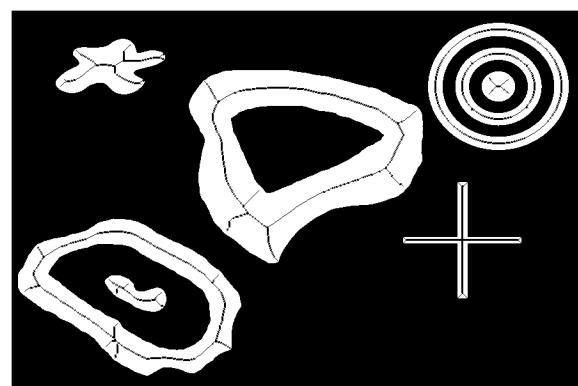


Figura f.74 - Imagen procesada (BW4)

Ejemplo 4: Calcular el esqueleto de dos microradiografías, la primera de un hueso y la segunda del mismo hueso al que se le ha aplicado ácido nítrico. Observar cómo se visualizan mejor las diferencias.

```
I = imread('A.tif'); imshow(I);
I2 = imread('B.tif'); figure, imshow(I2);

% Pasamos a imagen binaria con un umbral apropiado e igual para ambas.
```

```
level = 0.20;  
BW = im2bw(I, level);  
BW2 = im2bw(I2, level);  
  
% Calculamos el esqueleto.  
  
S = bwmorph(BW,'skel',Inf); figure, imshow(S)  
S2 = bwmorph(BW2,'skel',Inf); figure, imshow(S2)
```

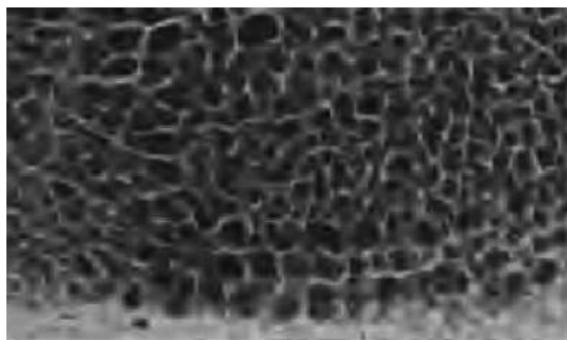


Figura f.75 - Imagen original (I) del hueso

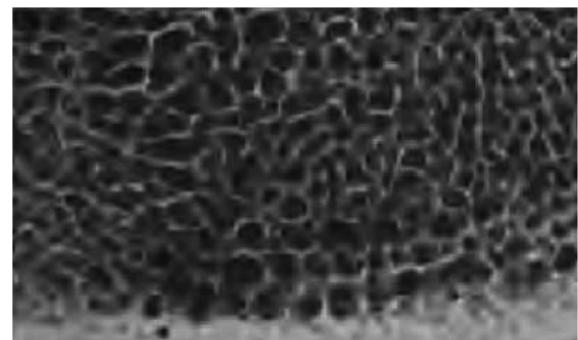


Figura f.76 - Imagen original (I2) del hueso tras aplicarle ácido

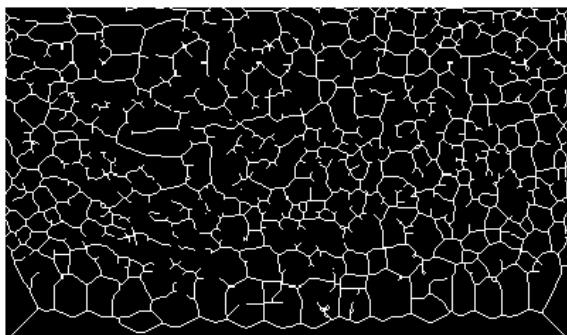


Figura f.77 - Esqueleto (S) de la imagen original I

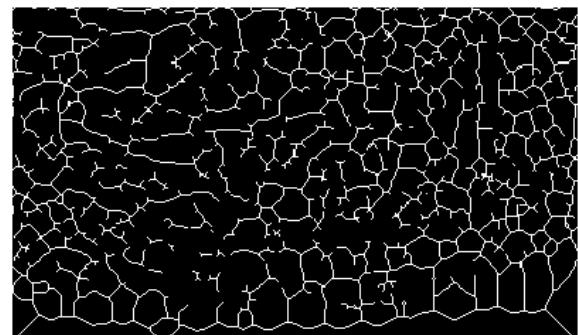


Figura f.78 - Esqueleto (S2) de la imagen original I2

Funciones relacionadas

bweuler, bwperim, imbothat, imclose, imdilate, imerode, imfill, imopen, imtophat

bwpack

Empaquetado de una imagen binaria

Introducción

El empaquetado de imágenes binarias se utiliza para acelerar algunas operaciones morfológicas como la dilatación o la erosión. Si el parámetro de entrada de las funciones `imdilate` o `imerode` es una imagen binaria empaquetada, la función usa una rutina especial para realizar la operación más rápido (básicamente el código en C o C++ opera con múltiples píxeles a la vez, resultando en un procesado más rápido).

Sintaxis

```
BWP = bwpack(BW)
```

Descripción

`BWP = bwpack(BW)` empaqueta una imagen binaria `BW` en formato `uint8` en un array `BWP` en formato `uint32` al que se le llama imagen binaria empaquetada. Como cada pixel en 8 bits solo puede valer 1 o 0, `bwpack` puede guardar cada pixel en un único bit de la imagen empaquetada de salida.

`bwpack` procesa los píxeles de la imagen por columnas, asignando grupos de 32 píxeles a los bits de un dato `uint32`. Grupos de 32 píxeles se guardan en números enteros de 32-bits.

`BW` puede ser del tipo `numeric` o `logical`, y debe ser 2-D, real, y no disperso. `BWP` es de tipo `uint32`.

`bwunpack` se usa para desempaquetar imágenes binarias empaquetadas.

Ejemplos de uso

Ejemplo 1: Empaquetar, dilatar y desempaquetar una imagen binaria.

```
BW = imread('texto.tif');
imshow(BW)

% Empaquetar, dilatar y desempaquetar usando una diagonal como
estructura.
```

```
BWP = bwpack(BW);  
se = strel(eye(9), rot90(eye(9)));  
BWPD = imdilate(BWP, se, 'ispacked');  
BWD = bwunpack(BWPD);  
figure, imshow(BWD)
```

Funciones relacionadas

[bwunpack](#), [imdilate](#), [imerode](#)

bwperim

Cálculo de los píxeles del perímetro de los objetos de una imagen binaria

Sintaxis

```
BW2 = bwperim(BW1)
BW2 = bwperim(BW1, conn)
```

Descripción

`BW2 = bwperim(BW1)` devuelve una imagen binaria solo con los píxeles del perímetro de los objetos de la imagen de entrada `BW1`. Un pixel forma parte del perímetro si es distinto de cero y está conectado a al menos un cero (fondo). La conectividad por defecto es de 4 para dos dimensiones, 6 para tres dimensiones y `conn=ndims(BW), 'minimal'` para mayores dimensiones, siempre referidas al fondo de la imagen.

`BW2 = bwperim(BW1, conn)` especifica la conectividad. `conn` puede tener cualquiera de los siguiente valores escalares (siempre simétrica respecto su elemento central):

Valor	Significado
Conejividades Bidimensionales	
4	Vecindad de 4 píxeles: Un par de píxeles colindantes forman parte del mismo objeto solo si están activos y conectados en vertical u horizontal.
8	Vecindad de 8 píxeles: Un par de píxeles colindantes forman parte del mismo objeto solo si están activos. (Por defecto)
Conejividades Tridimensionales	
6	Vecindad de 6 píxeles.
18	Vecindad de 18 píxeles.
26	Vecindad de 26 píxeles. (Por defecto)
<p>Nota: La conectividad se puede definir para cualquier dimensión de forma más general utilizando para <code>conn</code> una matriz de 3-por-3-por...-por-3 de ceros y unos. Los unos definen las localizaciones de los píxeles vecinos relativos al elemento central de <code>conn</code>. Por ejemplo, <code>conn=ones(3)</code> define una conectividad de 8 píxeles.</p>	

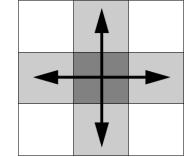


Figura f.32 - vecindad 4

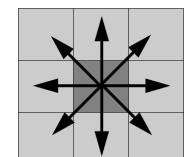


Figura f.33 - vecindad 8

`BW1` debe ser del tipo `numeric` o `logical`, y debe ser no disperso. `BW2` es de tipo `logical`.

Ejemplos de uso

Ejemplo 1: Visualizar cómo `bwperim` calcula los píxeles del perímetro encontrando los píxeles distintos de 0 conectados a píxeles igual a 0, para diferentes conectividades.

```
bw = [ 0      0      0      0      0      0      0
       0      0      1      1      1      0      0
       0      1      1      1      1      1      0
       0      1      1      1      1      1      0
       0      1      1      1      1      1      0
       0      0      1      1      1      0      0
       0      0      0      0      0      0      0 ]
```

```
% Perímetro para conectividad de 4: el pixel debe estar conectado a un cero por norte, sur, este u oeste.
```

```
perim4 = bwperim(bw, 4)
```

```
% Perímetro para conectividad de 8: el pixel debe estar conectado a un cero por norte, noreste, este, sureste, sur, suroeste, oeste o noroeste.
```

```
perim8 = bwperim(bw, 8)
```

```
% Perímetro para conectividad de 2 píxeles el superior e inferior: el pixel debe estar conectado a un cero por el norte o por el sur.
```

```
conn2 = [ 0 1 0
          0 1 0
          0 1 0 ];
```

```
perim2 = bwperim(bw, conn2)
```

```
bw =
0      0      0      0      0      0      0
0      0      1      1      1      0      0
0      1      1      1      1      1      0
0      1      1      1      1      1      0
0      1      1      1      1      1      0
0      0      1      1      1      0      0
0      0      0      0      0      0      0
```

```
perim4 =
0      0      0      0      0      0      0
0      0      1      1      1      0      0
0      1      0      0      0      1      0
0      1      0      0      0      1      0
0      1      0      0      0      1      0
0      0      1      1      1      0      0
0      0      0      0      0      0      0
```

```
perim8 =
0      0      0      0      0      0      0
0      0      1      1      1      0      0
0      1      1      0      1      1      0
0      1      0      0      0      1      0
```

```

0      1      1      0      1      1      0
0      0      1      1      1      0      0
0      0      0      0      0      0      0

perim2 =
0      0      0      0      0      0      0
0      0      1      1      1      0      0
0      1      0      0      0      1      0
0      0      0      0      0      0      0
0      1      0      0      0      1      0
0      0      1      1      1      0      0
0      0      0      0      0      0      0

```

Notar que como la conectividad se define para los píxeles de fondo (valor 0), con una conectividad `conn = 8` se obtiene unos píxeles con conectividad 4 en el perímetro, mientras que para una conectividad `conn = 4` se obtiene unos píxeles con conectividad 8 en el perímetro.

Ejemplo 1: Encontrar los píxeles del perímetro de los objetos de una imagen.

```

BW1 = imread('circulos.tif');
BW2 = bwperim(BW1,8);
figure, imshow(BW1)
figure, imshow(BW2)

```

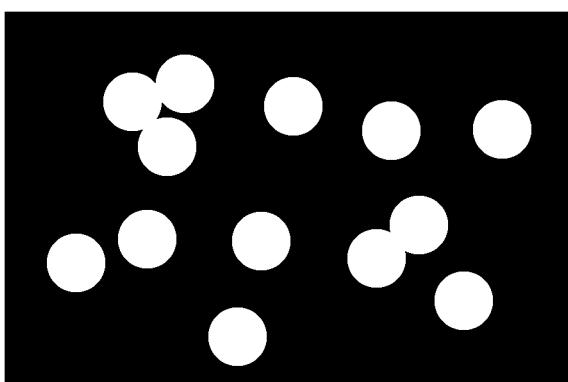


Figura f.48 - Imagen original (BW1)

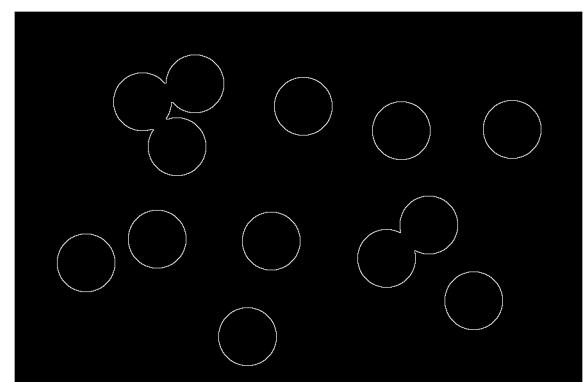


Figura f.79 - Imagen procesada (BW2) con bwperim

Funciones relacionadas

`bwarea`, `bwboundaries`, `bweuler`, `bwtraceboundary`, `conndef`, `imfill`

bwselect

Selección de objetos en una imagen binaria

Sintaxis

```
BW2 = bwselect(BW,c,r,n)
BW2 = bwselect(BW,n)
[BW2,idx] = bwselect(...)
BW2 = bwselect(x,y,BW,xi,yi,n)
[x,y,BW2,idx,xi,yi] = bwselect(...)
```

Descripción

`BW2 = bwselect(BW,c,r,n)` devuelve una imagen binaria con todos los objetos de la imagen binaria `BW` que se superponen al pixel de coordenadas `(r,c)`. `r` y `c` pueden ser escalares o vectores de igual longitud. Si `r` y `c` son vectores, `BW2` contiene el conjunto de objetos que se superponen a cualquiera de los píxeles `(r(k),c(k))`. `n` puede valer 4 o 8 (por defecto), donde 4 significa conectividad-4 y 8 significa conectividad-8. Los objetos son conjuntos de píxeles activos conectados.

`BW2 = bwselect(BW,n)` muestra la imagen `BW` en pantalla permitiendo que se puedan seleccionar las coordenadas `(r,c)` con el ratón. Si se omite `BW`, `bwselect` opera en a imagen con los ejes en uso. Los puntos se añaden haciendo clic con el ratón y se borran usando las teclas de retroceso o borrado. El último punto se añade con un clic derecho o un doble clic; presionando **retroceso** se finaliza la selección si añadir ningún punto.

`[BW2,idx] = bwselect(...)` devuelve los índices lineales del pixel correspondiente a los objetos seleccionados.

`BW2 = bwselect(x,y,BW,xi,yi,n)` usa los vectores `x` e `y` para establecer un sistema de coordenadas para `BW1`. `xi` e `yi` son escalares o vectores de igual longitud que especifican la localización en este sistema de coordenadas.

`[x,y,BW2,idx,xi,yi] = bwselect(...)` devuelve los parámetros `XData` e `YData` en `x` e `y`, la imagen de salida en `BW2`, los índices lineales para todos los píxeles pertenecientes a los objetos seleccionados en `idx`, y las coordenadas espaciales especificadas en `xi` e `yi`.

Si se llama a `bwselect` sin argumentos de salida, la imagen resultante se muestra en una nueva figura.

La imagen de entrada `BW` puede ser del tipo `numeric` o `logical` y debe ser 2-D y no dispersa. La imagen de salida `BW2` es de tipo `logical`.

Ejemplos de uso

Ejemplo 1: Seleccionar objetos a partir de coordenadas en una imagen binaria.

```
BW1 = imread('texto.tif');
imshow(BW1);
c = [316 356];
r = [69 141];
BW2 = bwselect(BW1,c,r,4);
figure, imshow(BW2)
```



Figura f.80 - Imagen original (BW1)

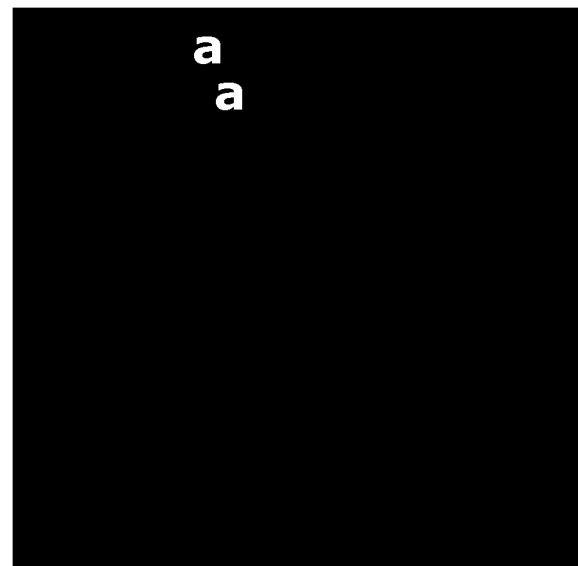


Figura f.81 - Imagen procesada (BW2) con bwselect

Funciones relacionadas

`bwlabel`, `imfill`, `impixel`, `roipoly`, `roifill`

bwtraceboundary

Cálculo de la frontera de un objeto en una imagen binaria

Sintaxis

```
B = bwtraceboundary(BW,P,fstep)
B = bwtraceboundary(BW,P,fstep,conn)
B = bwtraceboundary(...,N,dir)
```

Descripción

`B = bwtraceboundary(BW, P, fstep)` calcula la línea exterior de un objeto en la imagen binaria `BW` empezando desde las coordenadas en fila y columna del punto en la frontera especificado en el vector de dos elementos `P`. Los píxeles distintos de cero forman parte de un objeto y los píxeles de valor 0 constituyen el fondo. A diferencia de la función `bwboundaries`, que calcula todas las fronteras de todos los objetos y huecos, `bwtraceboundary` solo calcula la frontera de un objeto.

`fstep` es una cadena que especifica la dirección de búsqueda inicial para el siguiente pixel de objeto conectado a `P`. Se usan cadenas como '`N`' para Norte o '`NE`' para Noreste, para especificar la dirección. La siguiente figura ilustra todos los posibles valores que puede tomar `fstep`.

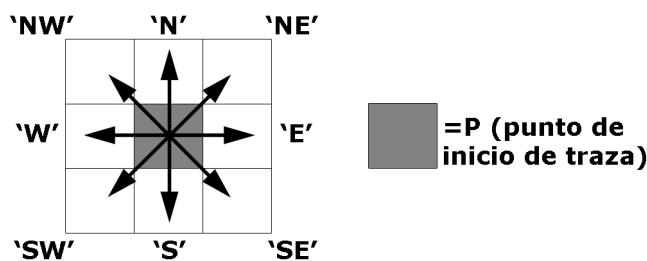


Figura f.82 - Valores de `fstep`

`bwtraceboundary` devuelve `B`, una matriz de `Q`-por-2 donde `Q` es el número de píxeles frontera de la región. `B` guarda las coordenadas fila y columna de los píxeles frontera.

`B = bwtraceboundary(bw, P, fstep, conn)` especifica la conectividad a usar cuando se traza la frontera. `conn` puede tener cualquiera de los siguientes valores escalares:

Valor	Significado
4	Vecindad de 4 píxeles. Nota: Con esta conectividad, <code>fstep</code> está limitado a los siguientes valores: ' <code>N</code> ', ' <code>E</code> ', ' <code>S</code> ', y ' <code>W</code> '.
8	Vecindad de 8 píxeles. (Por defecto)

`B = bwtraceboundary(. . . , N, dir)` especifica `n`, el número máximo de píxeles frontera, y `dir`, la dirección en la que trazar la frontera. Cuando `n` vale `Inf` (infinito) (por defecto) el algoritmo identifica todos los píxeles de la frontera. `dir` puede tener cualquiera de los siguientes valores:

Valor	Significado
'clockwise'	Búsqueda en dirección de las agujas del reloj. (Por defecto)
'counterclockwise'	Búsqueda en dirección contraria a las agujas del reloj.

`BW` puede ser del tipo `numeric` o `logical` y debe ser real, 2-D y no disperso. `B`, `P`, `conn` y `N` son de tipo `double`. `dir` y `fstep` son cadenas de caracteres.

Ejemplos de uso

Ejemplo 1: Dibujar y representar la frontera de un objeto en una imagen binaria dada mediante la función `bwtraceboundary` y luego dibujar y representar todas las fronteras mediante la función `bwboundaries`.

```
% Leemos y mostramos la imagen.

BW = imread('circulos.tif');
imshow(BW)

% Determinamos las coordenadas (fila y columna) del pixel distinto de
% cero de la frontera del objeto desde el que queramos que empiece a
% trazar. En nuestro caso el primer objeto de arriba a la izquierda, que
% está aproximadamente en 1/3 del ancho total.

dim = size(BW)
col = round(dim(2)/3);
row = min(find(BW(:,col)))

% Llamamos a bwtraceboundary para trazar la frontera del objeto empezando
% en ese punto. Usamos las opciones mínimas.

boundary = bwtraceboundary(BW,[row, col], 'N');
```

```
% Mostramos la imagen original y la frontera creada bwtraceboundary.  
  
figure, imshow(BW)  
hold on;  
plot(boundary(:,2),boundary(:,1),'g','LineWidth',3);  
  
% Para trazar las fronteras de todos los objetos en la imagen usamos la  
función bwboundaries que por defecto encuentra todas las fronteras de  
todos los objetos de la imagen, incluidos los objetos dentro de objetos.  
Para evitar huecos en negro dentro de los objetos principales que puedan  
ser interpretados como otros objetos, rellenamos los objetos principales  
con la función imfill.  
  
BW_filled = imfill(BW,'holes');  
boundaries = bwboundaries(BW_filled);  
  
% bwboundaries devuelve una matriz de celdas donde cada celda contiene  
las coordenadas (fila-columna) de un objeto en la imagen. Mostramos los  
bordes de todos los objetos usando estas coordenadas devueltas por la  
función.  
  
for k=1:10  
    b = boundaries{k};  
    plot(b(:,2),b(:,1),'g','LineWidth',3);  
end
```

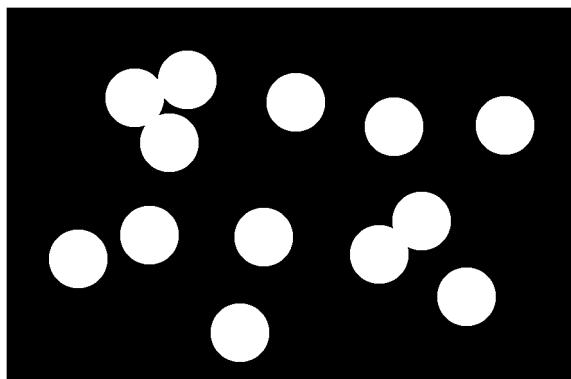


Figura f.48 - Imagen original (BW)

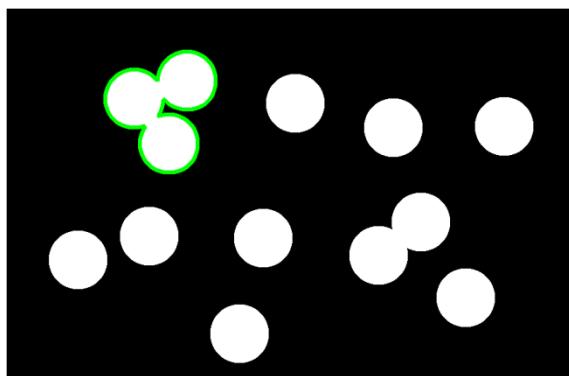


Figura f.83 - Imagen original (BW) con trazo de la
frontera de un objeto

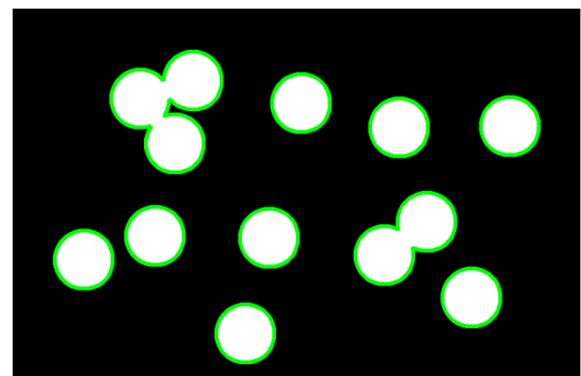


Figura f.84 - Imagen original (BW) con trazo de la
frontera de todos los objetos

Funciones relacionadas

bwboundaries, bwperim

bwulterode

Erosión última de una imagen binaria

Sintaxis

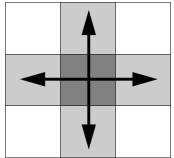
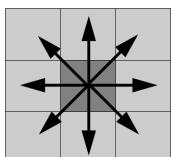
```
BW2 = bwulterode(BW)
BW2 = bwulterode(BW,method,conn)
```

Descripción

`BW2 = bwulterode(BW)` calcula la erosión última de la imagen binaria `BW`. La erosión última es el resultado de varios procesos de erosión que paran justo antes de eliminar los últimos píxeles aislados. La erosión última de `BW` consiste en el máximo regional (píxeles con un valor `h` tal que cada pixel en la vecindad del máximo tiene un valor menor que `h`) de la Transformada de distancia euclídea del complemento de `BW`.

La conectividad por defecto para el cálculo del máximo regional es 8 para 2 dimensiones, 26 para tres dimensiones y `conndef(ndims(BW), 'maximal')` para dimensiones mayores.

`BW2 = bwulterode(BW,method,conn)` especifica el método de la transformada de distancia y la conectividad del máximo regional. El método se define con la variable `method` y puede ser: '`euclidean`', '`cityblock`', '`chessboard`' y '`quasi-euclidean`'. La conectividad del máximo regional definida en `conn` puede ser cualquiera de las siguientes (siempre simétrica respecto su elemento central):

Valor	Significado
Conectividades Bidimensionales	
4	Vecindad de 4 píxeles: Un par de píxeles colindantes forman parte del mismo objeto solo si están activos y conectados en vertical u horizontal.
8	Vecindad de 8 píxeles: Un par de píxeles colindantes forman parte del mismo objeto solo si están activos. (Por defecto)
Conectividades Tridimensionales	
 <i>Figura f.32 - vecindad 4</i>	
 <i>Figura f.33 - vecindad 8</i>	

6	Vecindad de 6 píxeles.
18	Vecindad de 18 píxeles.
26	Vecindad de 26 píxeles. (Por defecto)

Nota: La conectividad se puede definir para cualquier dimensión de forma más general utilizando para `conn` una matriz de 3-por-3-por-...-por-3 de ceros y unos. Los unos definen las localizaciones de los píxeles vecinos relativos al elemento central de `conn`. Por ejemplo, `conn=ones(3)` define una conectividad de 8 píxeles.

`BW` puede ser del tipo `numeric` o `logical` y debe ser no disperso. Puede tener cualquier dimensión. El valor devuelto `BW2` es siempre un array tipo `logical`.

Ejemplos de uso

Ejemplo 1: Aplicar una erosión última a una imagen binaria.

```
BW = imread('cuadros.tif');
imshow(BW)
ultimateErosion = bwulterode(BW);
figure, imshow(ultimateErosion)
```

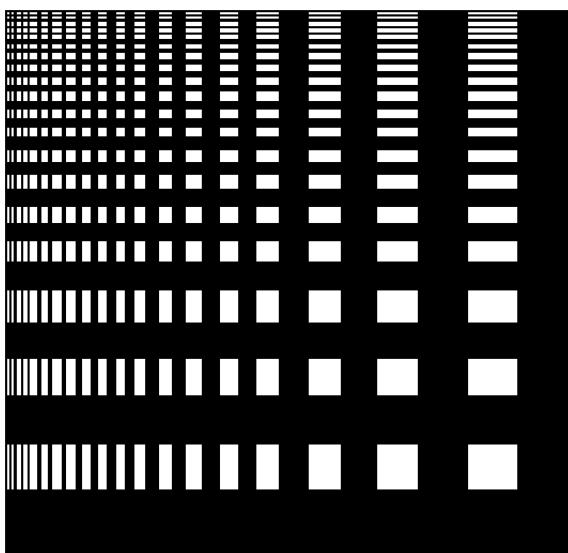


Figura f.50 - Imagen original (BW)

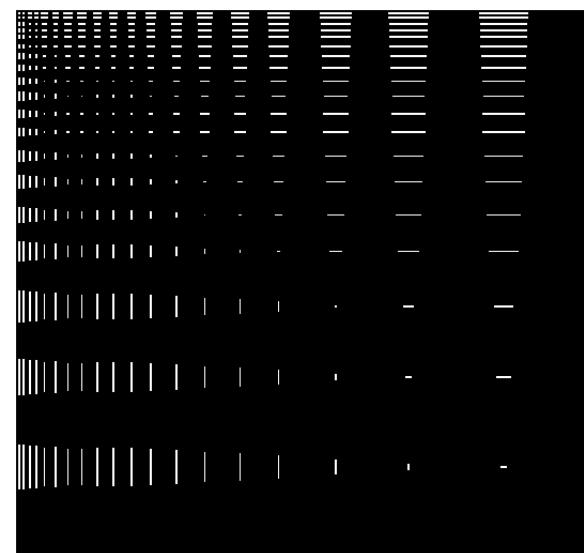


Figura f.85 - Imagen erosionada

Funciones relacionadas

`bwdist`, `conndef`, `imregionalmax`

bwunpack

Desempaquetado de una imagen binaria

Sintaxis

```
BW = bwunpack(BWP,M)
```

Descripción

`BW = bwunpack(BWP,M)` desempaquetá la imagen binaria `BWP`. `BWP` es un array en formato `uint32`. Cuando se desempaquetá `BWP`, `bwunpack` asigná el bit menos significativo de la primera fila de `BWP` al primer pixel en la primera fila de `BW`. El bit más significativo del primer elemento de `BWP` se asigná al primer pixel en la fila número 32 de `BW`, y así sucesivamente. `BW` tiene un tamaño de `M`-por-`N`, donde `N` es el número de columnas de `BWP`. Si se omite `M`, el valor por defecto es `32*size(BWP,1)`.

`bwpack` se utiliza para empaquetar imágenes binarias.

`BWP` es de tipo `uint32` y debe ser real, de 2-D y no dispersa. El valor devuelto por `BW` es de tipo `uint8`.

Ejemplos de uso

Ejemplo 1: Empaquetar, dilatar y desempaquetar una imagen binaria.

```
BW = imread('texto.tif');
imshow(BW)

% Empaquetar, dilatar y desempaquetar usando una diagonal como
% estructura.

BWP = bwpack(BW);
se = strel(eye(9), rot90(eye(9)));
BWPD = imdilate(BWP, se, 'ispacked');
BWD = bwunpack(BWPD);
figure, imshow(BWD)
```

Funciones relacionadas

`bwunpack`, `imdilate`, `imerode`

checkerboard

Creación de una imagen con patrón de tablero de damas

Introducción

La imagen generada por la función `checkerboard` es de gran utilidad si se utiliza como patrón de pruebas debido a que se puede redimensionar sin afectar a sus principales características y sobre ella se pueden visualizar muy bien ciertas operaciones como las transformaciones geométricas o las operaciones booleanas. También resulta de utilidad para visualizar zonas de una imagen superponiendo el patrón sobre la imagen.

Sintaxis

```
I = checkerboard  
I = checkerboard(n)  
I = checkerboard(n,p,q)
```

Descripción

`I = checkerboard` crea un cuadrado de 8-por-8 con patrón de tablero de damas y cuatro esquinas identificables. Cada cuadrado tiene 10 píxeles por lado. Los cuadrados claros en la mitad izquierda del tablero son blancos y los cuadrados claros en la mitad derecha del tablero son grises.

`I = checkerboard(n)` crea una imagen con patrón de tablero de damas donde cada cuadrado tiene `n` píxeles por lado.

`I = checkerboard(n,p,q)` crea una imagen rectangular con patrón de tablero de damas donde `p` especifica el número de filas (baldosas de 4 cuadrados) y `q` especifica el número de columnas (baldosas de 4 cuadrados). Si se omite `q`, se le da el valor de `p` por defecto y la imagen resultante es cuadrada.

Cada fila y columna está hecha de ‘baldosas’ (*tiles*) de cuatro cuadrados y `n` píxeles por lado, definida cada una como:

```
TILE = [DARK LIGHT; LIGHT DARK]
```



Figura f.61 -
Baldosa o ‘tile’

Ejemplos de uso

Ejemplo 1: Crear una imagen con patrón de tablero de damas donde el lado de cada cuadrado sea de 20 píxeles de longitud.

```
I = checkerboard(20);  
figure, imshow(I)
```

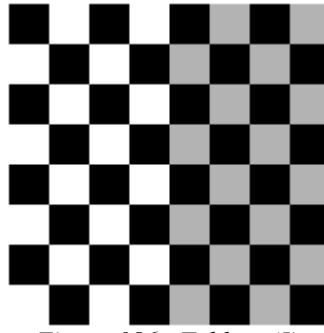


Figura f.86 - Tablero (I)

Ejemplo 2: Crear una imagen con patrón de tablero de damas con cuadrados de 10 píxeles de lado, 2 filas o baldosas de 4 cuadrados de alto y 3 de ancho.

```
J = checkerboard(10, 2, 3);  
figure, imshow(J)
```

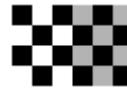


Figura f.87 - Tablero (J)

Ejemplo 3: Crear una imagen con patrón de tablero de damas de color blanco y negro.

```
K = (checkerboard > 0.5);  
figure, imshow(K)
```

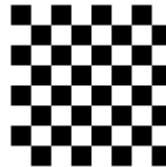


Figura f.88 - Tablero (K)

Ejemplo 4: Observar las diferencias entre la imagen de una mamografía antigua y la de una mamografía reciente superponiendo cada imagen en uno de los colores de un patrón de damas.

```
% Leemos las imágenes poniendo una encima de otra y mostramos las originales.
```

```
I = imread('A.tif');  
imshow(I);  
I2 = imread('B.tif');
```

```
hold on  
h = imshow(I2); % guardamos el puntero para luego.  
hold off  
figure, imshow(I); figure, imshow(I2)  
  
% Calculamos las filas y columnas de los cuadrados del patrón.  
  
[M,N] = size(I2);  
tambloques = 40;  
P = ceil(M / tambloques);  
Q = ceil(N / tambloques);  
  
% Creamos el patrón de damas rectangular de P filas (baldosas de 4  
cuadrados) por Q columnas (baldosas de 4 cuadrados) en blanco y negro.  
  
alpha_data = checkerboard(tambloques, P, Q) > 0;  
  
% El patrón creado tiene tamaño mayor que la imagen porque usa cuatro  
cuadrados por baldosa, así que lo reducimos al tamaño de la imagen.  
  
alpha_data = alpha_data(1:M, 1:N);  
  
% Superponemos cada imagen en un color del patrón utilizando la propiedad  
de transparencia 'AlphaData' utilizando las posiciones del tablero.  
  
set(h, 'AlphaData', alpha_data);
```

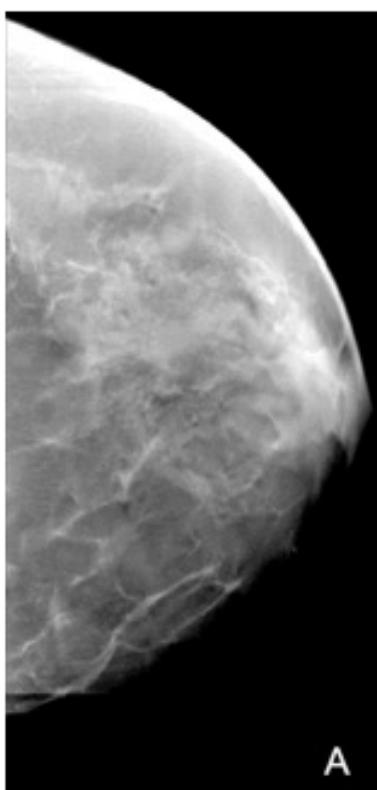


Figura f.89 - Imagen original (I)
[10]

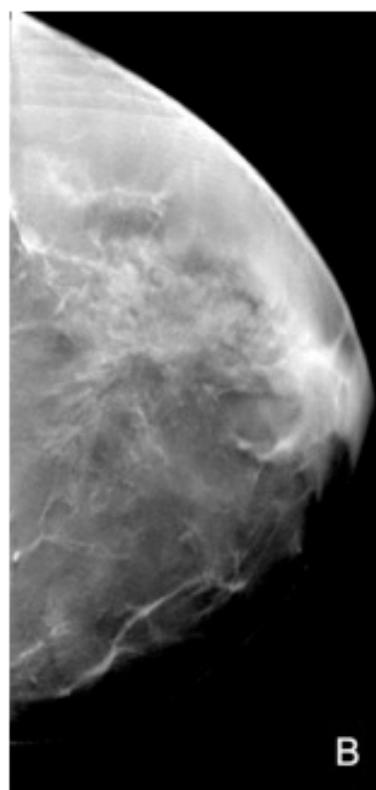


Figura f.90 - Imagen original (I2)

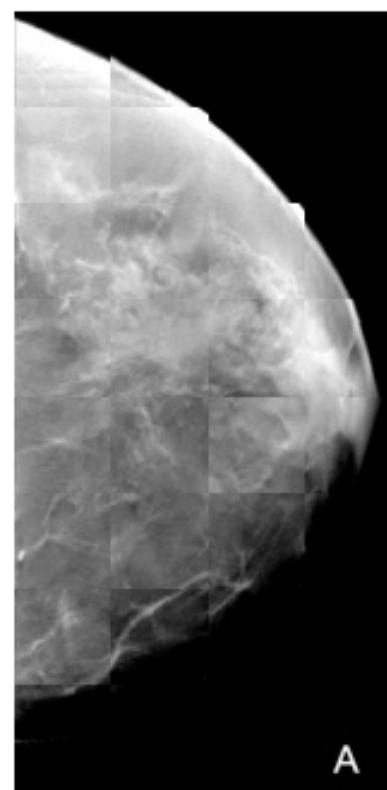


Figura f.91 - Imagen procesada
combinada de I e I2

Funciones relacionadas

cp2tform, imtransform, maketform

adapter.close

Cerrado de un objeto de clase ImageAdapter

Introducción

`ImageAdapter` es una clase abstracta que especifica la interfaz para la lectura y escritura de archivos por bloques o regiones, definiendo los métodos que la función `blockproc` usará para leer y escribir imágenes en el disco. Aunque la función `blockproc` solo admite determinados tipos de archivos, se puede conseguir trabajar con otro tipo de imágenes construyendo para ellos una clase que herede de la clase `ImageAdapter` y encapsule el código necesario.

Las clases `ImageAdapter` (para cada tipo de archivo) que heredan de la superclase `ImageAdapter` deben implementar obligatoriamente los métodos `readRegion` (lectura de una región de la imagen) y `close` (cerrar objeto `ImageAdapter`) para permitir una lectura básica de imágenes.

Sintaxis

```
adapter.close
```

Descripción

`adapter.close` cierra el objeto `ImageAdapter` y realiza cualquier operación de limpieza necesaria como el cierre de los descriptores de archivo. Cuando se construye una clase que hereda de la clase `ImageAdapter` se debe implementar este método.

col2im

Reordenación de las columnas de una matriz en bloques

Introducción

La función `im2col` reordena bloques de una imagen en columnas, mientras que `col2im` realiza la operación complementaria. El uso de estas funciones es bastante confuso por lo que se suele intentar evitarlas utilizando otras funciones más avanzadas y específicas como `reshape`, `blockproc` o `nlfiltter`. Se recomienda consultar también la función `im2col`.

Sintaxis

```
A = col2im(B,[m n],[mm nn],'distinct')
A = col2im(B,[m n],[mm nn],'sliding')
```

Descripción

`A = col2im(B,[m n],[mm nn],'distinct')` reordena cada columna de `B` en un bloque distinto de `m`-por-`n` para crear la matriz `A` de tamaño `mm`-por-`nn`. Si `B = [A11(:) A21(:) A12(:) A22(:)]`, donde cada columna tiene la longitud de `m*n`, entonces `A = [A11 A12; A21 A22]` donde cada `Aij` es de tamaño `m`-por-`n`.

`A = col2im(B,[m n],[mm nn],'sliding')` reordena el vector fila `B` en una matriz de tamaño $(mm-m+1)$ -por- $(nn-n+1)$. `B` debe ser un vector de tamaño 1 -por- $(mm-m+1)*(nn-n+1)$, que es normalmente el resultado de procesar la salida de `im2col(...,'sliding')` con alguna función de compresión de columna (como `sum` o `mean` por ejemplo).

`col2im(B,[m n],[mm nn])` es lo mismo que `col2im(B, [m n], [mm nn], 'sliding')`.

`B` puede ser del tipo `numeric` o `logical`. El valor devuelto `A` será de la misma clase que `B`.

Ejemplos de uso

Ejemplo 1: Reordenar `B` en bloques distintos de 2-por-2 en una matriz de 4-por4.

```
A11 = [1; 2; 3; 4];
A21 = [5; 6; 7; 8];
A12 = [9; 10; 11; 12];
A22 = [13; 14; 15; 16];
```

```
B = [A11 A21 A12 A22]

A = col2im(B,[2 2],[4 4],'distinct')

B =
    1     5     9    13
    2     6    10    14
    3     7    11    15
    4     8    12    16

A =
    1     3     9    11
    2     4    10    12
    5     7    13    15
    6     8    14    16

% Corresponde efectivamente a A = [A11 A12;A21 A22].
```

Ejemplo 2: Reorganizar un array en columnas con la función `im2col` y recuperar el array original con la función inversa.

```
A = [1:10;11:20;21:30;31:40]
B = im2col(A,[2,5],'distinct')
C = col2im(B,[2,5],[4,10],'distinct')

A =
    1     2     3     4     5      6     7     8     9     10
    11    12    13    14    15     16    17    18    19    20
    21    22    23    24    25     26    27    28    29    30
    31    32    33    34    35     36    37    38    39    40

B =
    1     21     6     26
    11    31    16     36
    2     22     7     27
    12    32    17     37
    3     23     8     28
    13    33    18     38
    4     24     9     29
    14    34    19     39
    5     25    10     30
    15    35    20     40

C =
    1     2     3     4     5      6     7     8     9     10
    11    12    13    14    15     16    17    18    19    20
    21    22    23    24    25     26    27    28    29    30
    31    32    33    34    35     36    37    38    39    40
```

Funciones relacionadas

`blockproc`, `colfilt`, `im2col`, `nlfiltter`

colfilt

Operaciones de vecindad en columnas - filtrado espacial no lineal

Introducción

Las funciones `colfilt` y `nlfiltter` se utilizan principalmente para aplicar un filtrado espacial no lineal. Este filtrado se basa en operaciones de vecindad y a diferencia del filtrado espacial lineal, que calcula la suma de productos (operación lineal), este utiliza operaciones no lineales que involucran a los píxeles de una vecindad. Por ejemplo, el hacer que la respuesta a cada punto sea igual al máximo valor en su vecindad es una operación de filtrado no lineal.

La función `nlfiltter` realiza operaciones directamente en 2 dimensiones mientras que `colfilt` organiza los datos en forma de columnas. `colfilt` necesita más memoria para operar pero lo hace de forma más rápida que `nlfiltter`, por lo que suele ser la elegida en la mayoría de aplicaciones de procesado digital de imágenes. Solo se suele prescindir de `nlfiltter` cuando el filtrado puede aplicarse alternativamente con funciones específicas como por ejemplo `imfilter`, `ordfilt2` o `spfilt` que consumen mucha menos memoria.

Sintaxis

```
B = colfilt(A,[m n],block_type,fun)
B = colfilt(A,[m n],[mblock nblock],block_type,fun)
B = colfilt(A,'indexed',...)
```

Descripción

`B = colfilt(A,[m n],block_type,fun)` procesa la imagen `A` recolocando cada bloque de `m`-por-`n` en una columna de una matriz temporal según el modo especificado en `block_type` y aplicándole después la función `fun` que debe ser una referencia o puntero a una función (*function handle*). Si es necesario se usa la función `colfilt` para llenar con ceros la matriz `A`.

Antes de llamar a `fun`, `colfilt` llama a `im2col` para crear la matriz temporal. Después, `colfilt` recoloca las columnas de la matriz de nuevo en `m`-por-`n` bloques usando `col2im`.

`block_type` es una cadena que puede tener uno de los siguientes valores:

Valor	Descripción
'distinct'	Recoloca cada bloque distinto de $m \times n$ de A en una columna de una matriz temporal y después le aplica la función fun que debe devolver una matriz del mismo tamaño que la matriz temporal. Luego colfilt recoloca las columnas de la matriz devuelta por fun en $m \times n$ bloques diferentes.
'sliding'	Usa una operación de vecindad tipo 'sliding' que consiste en coger sucesivamente bloques rectangulares de $m \times n$ píxeles para cada pixel de entrada y posteriormente obtener el pixel de salida aplicando una función a los píxeles vecinos. Recoloca cada bloque de $m \times n$ de A en una columna de una matriz temporal y después le aplica la función fun. fun debe devolver un vector fila que contenga un valor individual para cada columna de la matriz temporal. (Funciones de compresión de columnas como sum devuelven el tipo de salida apropiado). colfilt finalmente recoloca el vector devuelto por fun en una matriz del mismo tamaño que A.

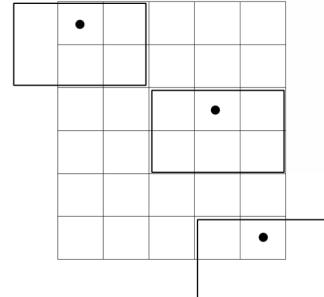


Figura f.92 - Bloques 'sliding' para operaciones de vecindad tipo 'sliding'

B = colfilt(A,[m n],[mblock nblock],block_type,fun) procesa la matriz A pero en bloques de tamaño mblock-por-nblock para ahorrar memoria. Notar que usando el argumento [mblock nblock] no cambia el resultado de la operación.

B = colfilt(A,'indexed',...) procesa A como una imagen indexada, rellenando con 0's si la clase de A es uint8 o uint16, o 1's si la clase de A es double o single.

Para evitar los problemas de bordes inherentes al filtrado espacial se debe llenar con ceros la imagen de entrada antes del filtrado. La mejor forma de hacerlo es mediante la función padarray que para funciones en 2 dimensiones tiene la siguiente sintaxis: fp = padarray(f, [r c], method, direction) donde f es la imagen de entrada, fp la de salida rellenada, y las variables method y direction definen la forma y dirección de relleno. A continuación se incluye un ejemplo utilizando esta función.

Nota: Para ahorrar memoria, la función colfilt podría dividir A en sub-imágenes y procesar una cada vez. Esto implica que se llamaría varias veces a fun y que el primer argumento de fun tendría diferente número de columnas cada vez.

colfilt opera de forma similar pero más rápido que blockproc y nlfilter.

La imagen de entrada *A* puede ser de cualquier clase que soporte *fun*. La clase de *B* depende de la clase de la salida de *fun*.

Ejemplos de uso

Ejemplo 1: Asignar a cada pixel de salida el valor medio de la vecindad de 5-por-5 del pixel (operación de vecindad tipo ‘sliding’).

```
I = imread('rueda.tif');
figure, imshow(I)
I2 = uint8(colfilt(I,[5 5],'sliding',@mean));
figure, imshow(I2)
```



Figura f.93 - Imagen original (*I*)



Figura f.94 - Imagen procesada (*I2*)

Ejemplo 1: Comparar el resultado de aplicar un filtrado espacial no lineal a una imagen utilizando relleno y sin utilizarlo. La respuesta a este filtro en cualquier punto debe ser la media geométrica de la intensidad de los píxeles de la vecindad correspondiente (producto de los valores de intensidad en la vecindad elevados a $1/mn$ donde $m \times n$ es el tamaño de la vecindad).

```
% Creamos primero la función de filtrado con el nombre gmean en un archivo gmean.m aparte.
```

```
function v = gmean(A)
mn = size(A, 1); % la longitud de las columnas de A siempre es mn.
v = prod(A, 1).^(1/mn);
```

```
% Ejecutamos el filtrado llamando a la función desde la línea de comandos.
```

```
I = im2double(imread('rueda.tif')); imshow(I)
P = colfilt(I, [5 5], 'sliding', @gmean); figure, imshow(P)
```

% Repetimos pero rellenando la imagen para evitar errores en los bordes.

```
I2 = padarray(I, [5 5], 'replicate');
P2 = colfilt(I2, [5 5], 'sliding', @gmean); figure, imshow(P2)
```



Figura f.93 - Imagen original (I)



Figura f.95 - Imagen procesada (P)



Figura f.96 - Imagen procesada (P2)

No se aprecia muy bien la diferencia entre P y P2 pero la primera tiene un borde negro que se come parte de la imagen mientras la segunda al tener un borde (relleno) de margen antes de ser procesada, el borde del procesado le cae en el borde de margen y no en la imagen, sin pérdida de información. Esto es similar a lo que ocurre en el filtrado espacial lineal aunque para evitarlo se puede hacer desde la propia función (`imfilter`).

Funciones relacionadas

`blockproc`, `col2im`, `function_handle`, `im2col`, `nlfiltre`

conndef

Creación de un array de conectividades

Sintaxis

```
conn = conndef(num_dims,type)
```

Descripción

conn = conndef(num_dims,type) devuelve el array de conectividades de dimensión num_dims definido por type. Los valores que puede tomar type son los siguientes:

Valor	Descripción
'minimal'	Define una vecindad cuyos vecinos están tocando el elemento central de una superficie de dimensión (N-1) para el caso de dimensión N.
'maximal'	Define una vecindad que incluye a vecinos que tocan el elemento central de cualquier manera.

Varias funciones de la *toolbox* para el Procesado de Imágenes usan esta función para crear la conectividad por defecto del argumento de entrada.

Ejemplos de uso

Ejemplo 1: Obtener la conectividad mínima y máxima para un array de dos dimensiones:

```
% Conectividad mínima. Vemos como incluye a los vecinos que están tocando  
el elemento central a lo largo de una línea.
```

```
Connmin = conndef(2,'minimal')
```

```
% Conectividad máxima. Vemos como incluye a todos los vecinos que tocan  
al elemento central.
```

```
Connmax = conndef(2,'maximal')
```

```
Connmin =  
0 1 0  
1 1 1  
0 1 0
```

```
Connmax =  
1 1 1  
1 1 1  
1 1 1
```

convmtx2

Matriz de convolución en 2-D

Sintaxis

```
T = convmtx2(H,m,n)
T = convmtx2(H,[m n])
```

Descripción

`T = convmtx2(H,m,n)` devuelve la matriz de convolución `T` para la matriz `H`. Si `x` es una matriz de `m`-por-`n`, entonces `reshape(T*X(:,),size(H)+[m n]-1)` es lo mismo que `conv2(X,H)`.

`T = convmtx2(H,[m n])` devuelve la matriz de convolución donde las dimensiones `m` y `n` son un vector de dos elementos.

Las variables de entrada son de tipo `double`. La matriz de salida `T` es de tipo `sparse` o disperso. El número de elementos distintos de cero en `T` no es mayor que `prod(size(H))*m*n`.

Esta matriz se puede usar en una multiplicación de matrices/vectores para realizar una convolución en 2 dimensiones.

Ejemplo 1: Calcular la convolución de `A` y `B` de forma directa e indirecta.

```
% Calculamos la convolución de A y B.
A = rand(10); B = rand(10);
C = conv2(A,B);

% Sacamos la matriz de convolución de A y comprobamos que reordenando
obtenemos la misma convolución que antes.

D = convmtx2(A,10,10);
E = reshape(D*reshape(B,100,1),19,19);
isequal(C,E)

ans =
    1
```

Funciones relacionadas

`conv2`

corner

Búsqueda de los puntos esquina de una imagen

Introducción

Las esquinas son la forma más fiable de encontrar la correspondencia (puntos homólogos) entre imágenes. Su definición la podemos hacer a partir de tres píxeles, uno dentro de la imagen, otro en el borde y otro en la esquina. Si un pixel se encuentra dentro de un objeto, sus alrededores (cuadrado sólido) se corresponden con los alrededores de sus vecinos (cuadrado punteado), en cualquier dirección. Si un pixel se encuentra en el borde de un objeto, sus alrededores difieren de los de sus vecinos en una dirección mientras que coinciden en otra (la perpendicular). Un pixel esquina tiene alrededores diferentes en todos sus vecinos en cualquier dirección.

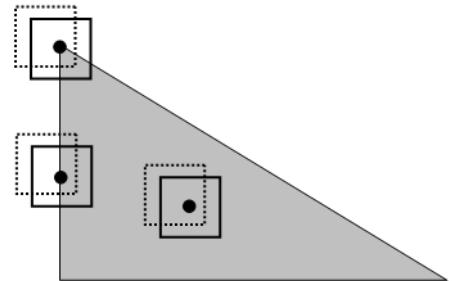


Figura f.68 - Definición de esquina

Las funciones `corner` y `cornermetric` detectan las esquinas de una imagen. En mayoría de aplicaciones es mejor usar la función `corner` para encontrar las esquinas en un solo paso. Si se quiere más control sobre la selección de esquinas es mejor usar la función `cornermetric` que calcula una matriz de medida de esquinas y después se puede utilizar un algoritmo propio para encontrar los valores más altos.

Sintaxis

```
C = corner(I)
C = corner(I, method)
C = corner(I, N)
C = corner(I, method, N)
C = corner(..., Name, Valor)
```

Descripción

`C = corner(I)` detecta las esquinas en la imagen `I` y devuelve su posición en la matriz `C` que almacena sus coordenadas `x` e `y`.

`C = corner(I, method)` detecta las esquinas en la imagen `I` utilizando el método especificado en la variable `method`. Los métodos que se pueden utilizar son:

Valor	Descripción
'Harris'	Detector de esquinas Harris. (Por defecto)
'MinimumEigenValue'	Método del mínimo valor eigen de Shi & Tomasi.

`C = corner(I, N)` detecta las esquinas en la imagen `I` y devuelve un máximo de `N` esquinas.

`C = corner(I, method, N)` detecta las esquinas utilizando el método especificado en la variable `method` y devuelve un máximo de `N` esquinas.

`C = corner(..., Name,Value)` especifica parámetros que permiten el control de varios aspectos del algoritmo de detección de esquinas. Se pueden definir pares de valores `Name,Value` donde `Name` es el nombre del argumento y `Value` es su valor correspondiente. Estos son los parámetros posibles:

Parámetro	Descripción
'FilterCoefficients'	Un vector <code>v</code> de coeficientes del filtro para el filtro de suavizado separable. El filtro completo se obtiene mediante el producto <code>v*v'</code> . Válido para los métodos ' <code>Harris</code> ' y ' <code>MinimumEigenValue</code> '. (Por defecto: <code>fspecial('gaussian',[5 1],1.5)</code>)
'QualityLevel'	Valor escalar <code>Q</code> donde $0 < Q < 1$ y que especifica la mínima calidad aceptable de las esquinas. Usar valores elevados de <code>Q</code> para evitar errores. (Por defecto: 0.01)
'SensitivityFactor'	Valor escalar <code>K</code> donde $0 < K < 0.25$ y que especifica el factor de sensibilidad a usar en el algoritmo de detección Harris. Cuanto menor sea el valor de <code>K</code> , el algoritmo detectará mejor las esquinas puntiagudas. Usar este parámetro solo con el método ' <code>Harris</code> '. (Por defecto: 0.04)

`I` es un array numérico no disperso. `C` es una matriz de `M-por-2` de tipo `double`.

Ejemplos de uso

Ejemplo 1: Encontrar y mostrar las esquinas en una imagen patrón de tablero de ajedrez.

```
I = checkerboard(50,2,2);
C = corner(I);
imshow(I)
hold on
plot(C(:,1), C(:,2), 'r*');
```

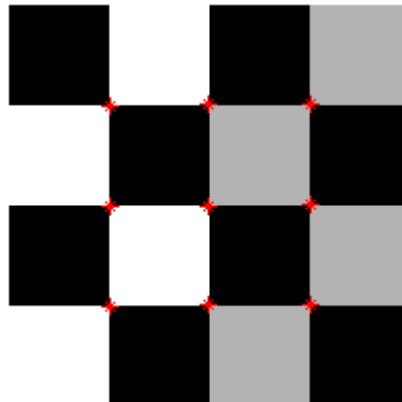


Figura f.97 - Esquinas encontradas

Funciones relacionadas

`cornermetric`

cornermetric

Creación de una matriz de medida de esquinas a partir de una imagen

Descripción

`CM = cornermetric(I)` genera una matriz de medida de esquinas para la imagen `I` en escala de grises o formato `logical`. La matriz `CM` se usa para detectar candidatos a esquinas en `I` y tiene el mismo tamaño que `I`. Los valores más altos en `CM` se corresponden con los píxeles en `I` con una mayor probabilidad de ser esquinas.

`CM = cornermetric(I, method)` genera una matriz de medida de esquinas para la imagen `I` en escala de grises o formato `logical` usando el método especificado en la variable `method`, que puede tomar los siguientes valores:

Valor	Descripción
'Harris'	Detector de esquinas Harris. (Por defecto)
'MinimumEigenValue'	Método del mínimo valor eigen de Shi & Tomasi.

`CM = cornermetric(..., param1, val1, param2, val2, ...)` genera una matriz de medida de esquinas para la imagen `I`, especificando parámetros y valores que controlan varios aspectos del algoritmo de cálculo de la matriz. Estos parámetros pueden ser:

Parámetro	Descripción
'FilterCoefficients'	Un vector <code>v</code> de coeficientes del filtro para el filtro de suavizado separable. El filtro completo se obtiene mediante el producto <code>v*v'</code> . Válido para los métodos 'Harris' y 'MinimumEigenValue'. (Por defecto: <code>fspecial('gaussian',[5 1],1.5)</code>)
'SensitivityFactor'	Valor escalar <code>k</code> donde $0 < k < 0.25$ y que especifica el factor de sensibilidad a usar en el algoritmo de detección Harris. Cuando menor sea el valor de <code>k</code> , el algoritmo detectará mejor las esquinas puntiagudas. Usar este parámetro solo con el método 'Harris'. (Por defecto: 0.04)

`I` es un array numérico no disperso. `CM` es una matriz de tipo `double`.

Ejemplos de uso

Ejemplo 1: Encontrar y mostrar las esquinas en una imagen patrón de tablero de ajedrez.

```
I = checkerboard(50,2,2);
imshow(I)

% Los picos de la matriz CM son las esquinas más probables. Calculamos
% los máximos regionales de la matriz CM que nos da los picos de los
% lugares más probables donde hay una esquina.

CM = cornermetric(I);
corner_peaks = imregionalmax(CM);

% Encontramos su posición.

[crow,ccol]= find(corner_peaks == true);

% Le damos color a los píxeles esquina y mostramos.

hold on
plot(crow, ccol, 'r*');
```

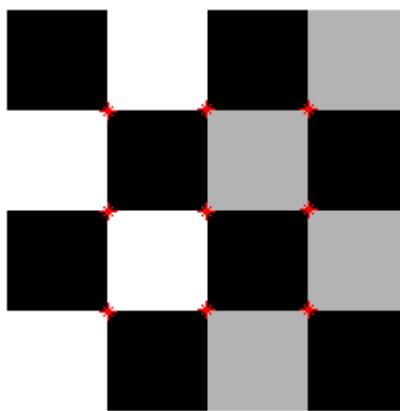


Figura f.97 - Esquinas encontradas

Funciones relacionadas

corner, edge, immovie, imshow

corr2

Coeficiente de correlación en 2-D

Sintaxis

```
r = corr2(A,B)
```

Descripción

`r = corr2(A,B)` calcula el coeficiente de correlación entre `A` y `B`, donde `A` y `B` son matrices o vectores del mismo tamaño.

`A` y `B` pueden ser tipo `numeric` o `logical`. El valor devuelto `r` es un escalar tipo `double`.

Ejemplos de uso

Ejemplo 1: Calcular el coeficiente de correlación entre una imagen y la misma imagen procesada con un filtro de mediana.

```
I = imread('pout.tif');
J = medfilt2(I);
R = corr2(I,J)

R =
    0.9959
```

Funciones relacionadas

`std2`, `corrcoef`

cp2tform

Deducción de la matriz de transformación a partir de pares de puntos de control

Introducción

El registro o alineación de imágenes es un proceso a través del cual se alinean dos o más imágenes de una misma escena compensando las aberraciones geométricas que se hayan podido generar. Existe una imagen base o de referencia y otra imagen distorsionada que se comparan para obtener la matriz de transformación y posteriormente aplicar la transformación geométrica para alinear ambas imágenes.

La IPT soporta el registro de imágenes basado en el uso de puntos de control, que son un subconjunto de píxeles cuyas localizaciones en ambas imágenes a alinear se pueden seleccionar de forma interactiva mediante la función `cpselect`. Una vez seleccionado un número suficiente de puntos, la función `cp2tform` se encarga de obtener la matriz de transformación que permitirá luego la alineación de las imágenes transformando inversamente la imagen distorsionada mediante la función `imtransform`.

Sintaxis

```
TFORM = cp2tform(input_points, base_points, transformtype)
TFORM = cp2tform(CPSTRUCT, transformtype)
[TFORM, input_points, base_points] = cp2tform(CPSTRUCT,...)
TFORM = cp2tform(..., 'polynomial', order)
TFORM = cp2tform(..., 'lwm', N)
[TFORM, input_points, base_points, input_points_bad, base_points_bad]=
cp2tform(..., 'piecewise linear')
```

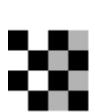
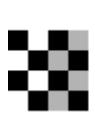
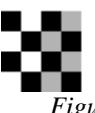
Descripción

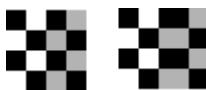
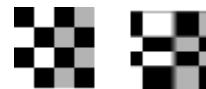
`TFORM = cp2tform(input_points, base_points, transformtype)` deduce una transformación a partir de pares de puntos de control y devuelve la transformación como una estructura `TFORM`. `input_points` es una matriz `m`-por-2 en formato `double` que contiene las coordenadas `x` e `y` de los puntos de control en la imagen distorsionada que se quiere transformar. `base_points` es una matriz de `m`-por-2 en formato `double` que contiene las coordenadas `x` e `y` de los puntos de control especificados en la imagen base o referencia. `transformtype` especifica el tipo de transformación espacial a deducir.

`TFORM = cp2tform(CPSTRUCT, transformtype)` trabaja en una estructura CPSTRUCT que contiene las matrices de puntos de control para la imagen de entrada y la imagen base. Se usa la herramienta de selección `cpselect` para crear la estructura CPSTRUCT.

`[TFORM, input_points, base_points] = cp2tform(CPSTRUCT,...)` devuelve los puntos de control que han sido usados en los `input_points` y `base_points`.

La siguiente tabla muestra las posibles transformaciones al usar `cp2tform` en orden creciente de complejidad:

Tipo de Transformación	Descripción	Puntos de Control mínimos	Ejemplo
'nonreflective similarity'	Usar esta transformación cuando las formas en la imagen de entrada no cambian pero la imagen está distorsionada con alguna combinación de traslación, rotación y escalado. Las líneas rectas se mantienen rectas y las paralelas se mantienen paralelas.	2 pares	  <i>Figura f.98 - Transformación nonreflective similarity</i>
'similarity'	Igual que 'nonreflective similarity' con la adición de una reflexión opcional.	3 pares	  <i>Figura f.99 - Transformación similarity</i>
'affine'	Usar esta transformación cuando las formas en la imagen de entrada exhiben deformación de cizalla. Las líneas rectas se mantienen rectas y las paralelas se mantienen paralelas pero los rectángulos se convierten en paralelogramos.	3 pares	  <i>Figura f.100 - Transformación affine</i>
'projective'	Usar esta transformación cuando la imagen aparezca inclinada. Las líneas rectas se mantienen rectas pero las líneas paralelas convergen hacia puntos de fuga que pueden o no caer dentro de la imagen.	4 pares	  <i>Figura f.101 - Transformación projective</i>

'polynomial'	Usar esta transformación cuando los objetos de la imagen están curvados. Cuando más grande sea el grado del polinomio mejor encajará, pero el resultado podrá tener más curvas que la imagen de referencia.	6 pares (orden 2) 10 pares (orden 3) 15 pares (orden 4)	 <i>Figura f.102 - Transformación polynomial</i>
'piecewise linear'	Usar esta transformación cuando haya partes en la imagen distorsionadas de forma diferente.	4 pares	 <i>Figura f.103 - Transformación piecewise linear</i>
'lwm' (<i>local weighted mean</i>)	Usar esta transformación cuando la distorsión varía localmente y la transformación 'piecewise linear' no es suficiente.	6 pares (recomendado 12)	 <i>Figura f.104 - Transformación lwm</i>

Sintaxis para transformaciones específicas:

TFORM = cp2tform(..., 'polynomial', order) devuelve una estructura TFORM especificando una transformación tipo 'polynomial' donde order es el orden del polinomio a usar, cuyo valor puede ser 2, 3 (por defecto), o 4.

TFORM = cp2tform(..., 'lwm', N) devuelve una estructura TFORM especificando una transformación tipo 'lwm' donde N especifica el número de puntos usados para deducir cada polinomio. Los N puntos más cercanos se usan para deducir un polinomio de orden 2 para cada par de punto de control. Si se omite N, por defecto es 12. N puede ser mínimo 6, pero hacerlo pequeño tiene riesgo de generar polinomios condicionados de forma incorrecta.

[TFORM, input_points, base_points, input_points_bad, base_points_bad]=cp2tform(..., 'piecewise linear') devuelve una estructura TFORM especificando una transformación tipo 'piecewise linear'. Devuelve los puntos de control usados en las variables input_points y base_points, y devuelve los puntos de control eliminados cuando no tienen un orden apropiado.

Ejemplos de uso

Ejemplo 1: Alinear una imagen original con la misma imagen a la que se la ha aplicado una rotación. Usamos `cp2tform` para obtener la matriz de transformación.

```
% Leemos las imagen original y la rotamos 10 grados para obtener la
% imagen de entrada distorsionada.

I1 = imread('calatrava.tif');
I2 = imrotate(I1, -10, 'bilinear');

% Introducimos manualmente los puntos de control (2 pares mínimo para la
% transformación elegida).

[input_points, base_points] = cpselect(I2, I1, 'Wait', true);

% Realizamos el Registro de la imagen que consiste en encontrar la
% transformación que haga pasar I2 a I1 usando los puntos de control.

t = cp2tform(input_points, base_points, 'nonreflective similarity');

% Transformamos I2 para que esté alineada con I1 rellenando la imagen con
% color blanco.

[II2] = imtransform(I2, t, 'FillValues', 255);

% Comparamos resultados.

figure('menu','none')
subplot(131), imshow(I1), title('I1')
subplot(132), imshow(I2), title('I2')
subplot(133), imshow(II2), title('I2 (alineada)')
```

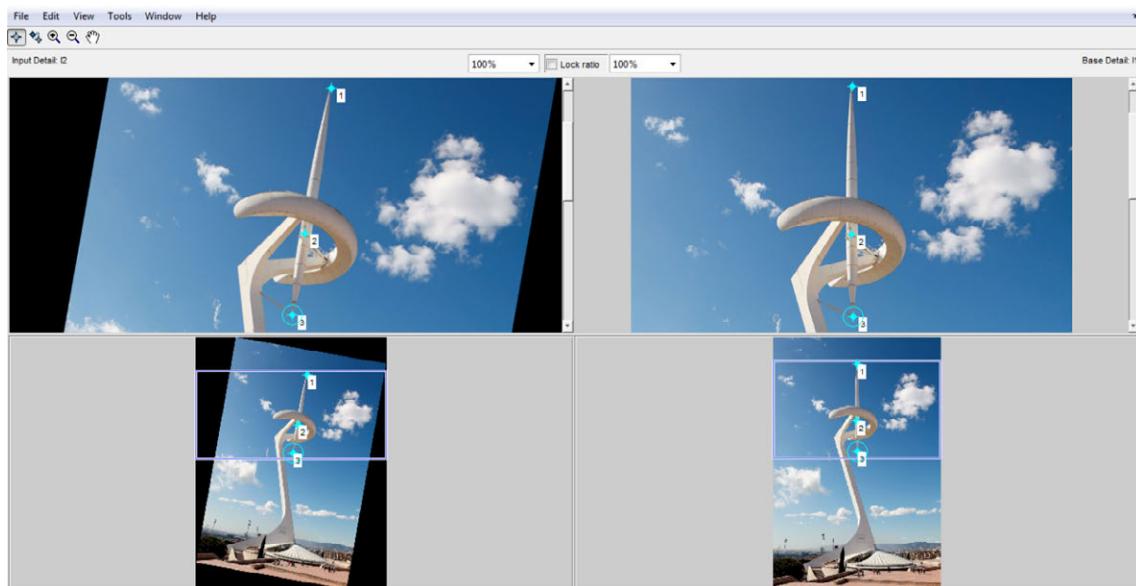


Figura f.105 - Detalle de puntos de control especificados

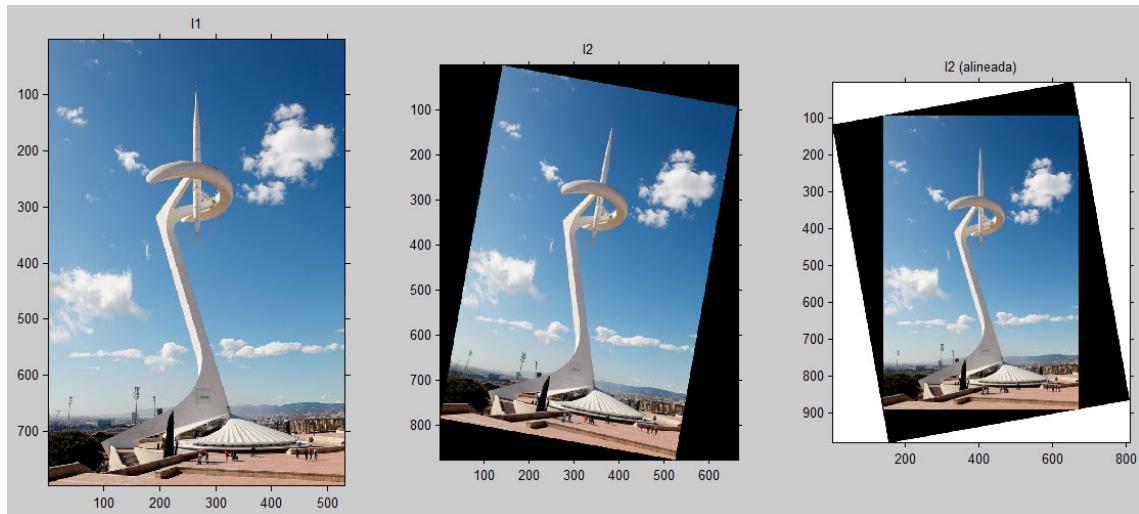


Figura f.106 - Imagen base (I1), distorsionada (I2) y alineada (I2)

Funciones relacionadas

cpcorr, cpselect, cpstruct2pairs, imtransform, tformfwd, tforminv

cpcorr

Afinado de la localización de los puntos de control mediante correlación cruzada

Sintaxis

```
input_points = cpcorr(input_points_in, base_points_in, input, base)
```

Descripción

`input_points = cpcorr(input_points_in, base_points_in, input, base)` usa una correlación cruzada normalizada para ajustar cada par de puntos de control de las imágenes de entrada y referencia `input` y `base` especificados en las variables `input_points_in` y `base_points_in`.

`input_points_in` debe ser una matriz `M`-por-2 de clase `double` que contiene las coordenadas de los puntos de control de la imagen de entrada especificada en la variable `input`. `base_points_in` es una matriz `M`-por-2 de clase `double` que contiene las coordenadas de los puntos de control de la imagen de entrada especificada en la variable `base`.

`cpcorr` devuelve los puntos de control ajustados en la variable `input_points`, una matriz de tipo `double` del mismo tamaño que `input_points_in`. Si `cpcorr` no puede hacer la correlación de una pareja de puntos de control, `input_points` contendrá las mismas coordenadas que `input_points_in` para ese par.

`cpcorr` solo modifica la posición de los puntos de control en un máximo de 4 píxeles. `cpcorr` es fiable en la selección y adquisición de los puntos de control pero las imágenes de entrada y referencia `input` y `base` deben tener la misma escala para que `cpcorr` sea efectiva.

`cpcorr` no puede ajustar un punto si ocurre cualquiera de los siguientes sucesos:

- Los puntos están demasiado cerca del borde de cada imagen.
- Hay regiones de la imagen alrededor de los puntos que contienen valores `Inf` o `NaN` (infinitos o indeterminados)

- La región alrededor de un punto en la imagen de entrada tiene desviación típica igual a cero.
- Hay regiones alrededor de los puntos que están correladas deficientemente.

Las imágenes de entrada y de referencia `input` y `base` pueden ser del tipo `numeric` y deben tener valores finitos. Los pares de puntos de control son de tipo `double`.

Ejemplos de uso

Ejemplo 1: Utilizar `cpcorr` para ajustar los puntos de control seleccionados en una imagen RGB. Mostrar la diferencia de valores obtenidos en cada caso.

```
I1 = imread('calatrava.tif');
I2 = imrotate(I1, -10, 'bilinear');

% Introducimos manualmente los puntos de control (2 pares mínimo para la transformación elegida, seleccionamos 3 pares).

[input_points, base_points] = cpselect(I2, I1, 'Wait', true)

% Afinamos la selección pasando solo un canal de la imagen RGB.

input_points_adj = cpcorr(input_points,base_points, I2(:,:,1),I1(:,:,1))

input_points =
    388.2500    131.2500
    356.7500    306.7500
    342.7500    388.2500

base_points =
    266.7500    86.7500
    266.2500   264.7500
    268.7500   346.7500

input_points_adj =
    387.6500    131.4500
    356.7500    306.7500
    345.2500    387.7500
```

Funciones relacionadas

`cp2tform`, `cpselect`, `imtransform`, `normxcorr2`

cpselect

Herramienta de selección de puntos de control en imágenes

Introducción

La IPT soporta el registro o alineación de imágenes basado en el uso de puntos de control, que son un subconjunto de píxeles cuyas localizaciones en ambas imágenes se pueden seleccionar de forma interactiva mediante la función `cpselect`. Una vez seleccionado un número suficiente de puntos, la función `cp2tform` se encarga de obtener la matriz de transformación que permitirá luego la alineación de las imágenes transformando inversamente la imagen distorsionada mediante la función `imtransform`. Más información y ejemplos de este proceso en la página de la función `cp2tform`.

Sintaxis

```
cpselect(input, base)
cpselect(input, base, CPSTRUCT_IN)
cpselect(input, base, xyinput_in, xybase_in)
h = cpselect(input, base,...)
cpselect(...,param1, val1,...)
```

Descripción

`cpselect(input, base)` inicia la herramienta de selección de puntos de control, una interfaz de usuario que permite seleccionar puntos de control en dos imágenes relacionadas: imagen de entrada (normalmente distorsionada) e imagen base o referencia. La imagen de entrada o `input` es la imagen que necesita ser transformada para ponerla en el sistema de coordenadas de la imagen original o `base`. `input` y `base` pueden ser ambas variables que contengan imágenes en escala de grises, en color verdadero o imágenes binarias, o cadenas que identifiquen archivos que contengan esas imágenes. La herramienta de selección de puntos de control devuelve los puntos de control en una estructura `CPSTRUCT`.

`cpselect(input, base, CPSTRUCT_IN)` inicia `cpselect` con un conjunto inicial de puntos de control que se guardan en `CPSTRUCT_IN`. Esta sintaxis permite reiniciar `cpselect` al estado anterior de los puntos de control guardados en `CPSTRUCT_IN`.

`cpselect(input, base, xyinput_in, xybase_in)` inicia `cpselect` con un conjunto inicial de puntos de control guardados en las variables `xyinput_in` e `xybase_in` que son matrices m -por-2 con las coordenadas de las imágenes `input` y `base` respectivamente.

`h = cpselect(input, base, ...)` devuelve un puntero `h` a la herramienta. Se puede usar el comando `close(h)` para cerrar la herramienta desde la línea de comandos.

`cpselect(..., param1, val1, ...)` inicia `cpselect` especificando ciertos parámetros y valores que controlan varios aspectos de la herramienta. Los nombres de parámetro se pueden abbreviar y no hay sensibilidad a las mayúsculas. Estos parámetros incluyen:

Parámetro	Descripción
'Wait'	Escalar en formato <code>Logical</code> que controla si <code>cpselect</code> espera a que el usuario cierre la herramienta para finalizar la selección de puntos de control. Si se deja en <code>false</code> (por defecto) se puede ejecutar <code>cpselect</code> al mismo tiempo que se ejecutan otros programas en MATLAB. Si se pone a <code>true</code> , se debe finalizar la tarea de selección para continuar con la ejecución del programa y de cualquier otra cosa en MATLAB. Cuando 'Wait' se pone a <code>true</code> , <code>cpselect</code> devuelve los pares de puntos seleccionados, no un puntero a la herramienta: <code>[xyinput_out, xybase_out] = cpselect(..., 'Wait', true)</code> donde <code>xyinput_out</code> y <code>xybase_out</code> son matrices de P -por-2 que guardan las coordenadas de entrada y de referencia respectivamente.

Las imágenes pueden ser en escala de grises, color verdadero o binarias. Una imagen en escala de grises puede ser de tipo `uint8`, `uint16`, `int16`, `single`, o `double`. Una imagen en color verdadero puede ser `uint8`, `uint16`, `single`, o `double`. Una imagen binaria tiene que ser de tipo `logical`.

Ejemplos de uso

Ejemplo 1: Iniciar la herramienta de selección de puntos de control con una imagen original y otra rotada pasándole unos puntos de control determinados.

```
I = checkerboard;
J = imrotate(I,30);
base_points = [11 11; 41 71];
input_points = [14 44; 70 81];
cpselect(J, I, input_points, base_points);
```

Ejemplo 2: Iniciar la herramienta de selección de puntos de control con una imagen original y otra rotada, seleccionar los puntos de control y recogerlos en dos variables usando la función `cpstruct2pairs`.

```
I = checkerboard;
J = imrotate(I,30);
cpselect(J, I);
[input_points,base_points] = cpstruct2pairs(cpstruct)
```

Ejemplo 3: Utilizar `cpselect` especificando el parámetro 'wait' para que el programa se pare hasta que se termine la selección de puntos de control y se cierre la herramienta recibiendo los puntos de control en dos variables.

```
I1 = imread('calatrava.tif');
I1 = rgb2gray(I1);
I2 = imrotate(I1, -10, 'bilinear', 'crop');

[input_points, base_points] = cpselect(I2, I1, 'Wait', true)
input_points_adj = cpcorr(input_points,base_points, I2(:,:,1),I1(:,:,1))
```

Funciones relacionadas

`cpcorr`, `cp2tform`, `cpstruct2pairs`, `imtransform`

cpstruct2pairs

Conversión de la estructura CPSTRUCT a un par de puntos de control válidos

Sintaxis

```
[input_points, base_points] = cpstruct2pairs (CPSTRUCT)
```

Descripción

[input_points, base_points] = cpstruct2pairs (CPSTRUCT) recoge una estructura CPSTRUCT creada a partir de la función cpselect y devuelve el array de coordenadas de pares de puntos de control válidos en las variables input_points y base_points. cpstruct2pairs elimina puntos no emparejados y puntos previsibles.

Ejemplos de uso

Ejemplo 1: Iniciar la herramienta de selección de puntos de control con una imagen original y otra rotada, seleccionar los puntos de control y recogerlos en dos variables usando la función cpstruct2pairs.

```
I = checkerboard;
J = imrotate(I,30);
cpselect(J, I);
[input_points,base_points] = cpstruct2pairs(cpstruct)
```

Nota: En la herramienta de selección de puntos de control se deben salvar los puntos de control desde el menú **File-Export Points To Workspace** y posteriormente indicar que es una estructura de nombre cpstruct con todos los puntos de control.

Funciones relacionadas

[cp2tform](#), [cpselect](#), [imtransform](#)

dct2

Transformada de Coseno discreta en 2-D

Introducción

La Transformada de Coseno Discreta (DCT) representa una imagen como una suma de sinusoides de magnitudes y frecuencias oscilantes. La DCT tiene la propiedad de que para una imagen corriente, la mayoría de información o energía importante para su visualización se localiza solo en unos pocos coeficientes de la DCT. Por eso es la transformada más utilizada en los estándares de compresión de imágenes, como en la compresión JPEG.

En MATLAB existen dos formas de calcular la DCT con la *toolbox* de procesado de imágenes. La primera forma es mediante la función `dct2`, que usa un algoritmo basado en la Transformada rápida de Fourier (FFT) para cálculos más rápidos con entradas grandes. La segunda forma es mediante la matriz de transformación DCT, que se obtiene con la función `dctmtx` y puede ser más eficiente cuando se trabaja con entradas pequeñas, sobre todo si son cuadradas de 8-por-8 o 16-por-16, típicas dimensiones en el procesado por bloques. La Transformada de coseno discreta inversa se calcula mediante la función `idct2`.

Sintaxis

```
B = dct2(A)
B = dct2(A,m,n)
B = dct2(A,[m n])
```

Descripción

`B = dct2(A)` devuelve la transformada de coseno discreta bidimensional de `A`. La matriz `B` es del mismo tamaño que `A` y contiene los coeficientes $B(k_1, k_2)$ de la transformada de coseno discreta.

`B = dct2(A,m,n)` rellena la matriz `A` con ceros hasta el tamaño `m`-por-`n` antes de realizar la transformación. Si `m` o `n` es más pequeño que la dimensión correspondiente de `A`, `dct2` trunca `A`.

`B = dct2(A,[m n])` igual que la sintaxis anterior.

A puede ser del tipo `numeric` o `logical`. La matriz devuelta B es de clase `double`.

Ejemplos de uso

Ejemplo 1: Calcular la transformada de coseno discreta de una imagen y observar dónde se concentra la mayor parte de la energía. A partir de esta observación se podría comprimir la imagen eliminando la información que tiene mayor energía y por tanto es la más redundante (situada en la esquina superior izquierda).

```
% Leemos una imagen en color y la convertimos a escala de grises.
```

```
RGB = imread('autumn.tif');  
imshow(RGB);  
I = rgb2gray(RGB);
```

```
% Sacamos la DCT de la imagen y la representamos.
```

```
J = dct2(I);  
figure, imshow(log(abs(J)),[]), colormap(jet(64)), colorbar;
```



Figura f.107 - Imagen original (RGB)

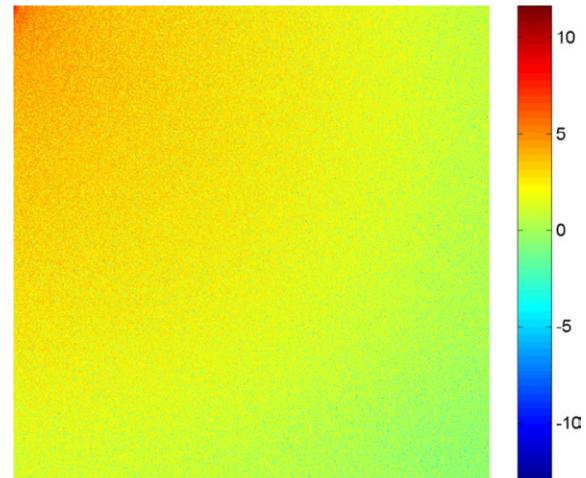


Figura f.108 - Energía de la imagen procesada J

Funciones relacionadas

`fft2`, `idct2`, `ifft2`

dctmtx

Matriz DCT (Transformada de coseno discreta)

Introducción

La Transformada de Coseno Discreta (DCT) representa una imagen como una suma de sinusoides de magnitudes y frecuencias oscilantes. La DCT tiene la propiedad de que para una imagen corriente, la mayoría de información o energía importante para su visualización se localiza solo en unos pocos coeficientes de la DCT. Por eso es la transformada más utilizada en los estándares de compresión de imágenes, como en la compresión JPEG.

En MATLAB existen dos formas de calcular la DCT con la *toolbox* de procesado de imágenes. La primera forma es mediante la función `dct2`, que usa un algoritmo basado en la Transformada rápida de Fourier (FFT) para cálculos más rápidos con entradas grandes. La segunda forma es mediante la matriz de transformación DCT, que se obtiene con la función `dctmtx` y puede ser más eficiente cuando se trabaja con entradas pequeñas, sobre todo si son cuadradas de 8-por-8 o 16-por-16, típicas dimensiones en el procesado por bloques. La Transformada de coseno discreta inversa se calcula mediante la función `idct2`.

Sintaxis

`D = dctmtx(n)`

Descripción

`D = dctmtx(n)` devuelve la matriz DCT de tamaño n-por-n.

Para una matriz `A` de n-por-n, `D*A` será una matriz de n-por-n cuyas columnas contendrán la DCT unidimensional de las columnas de `A`. `D'*A` es la DCT unidimensional inversa de las columnas de `A`.

La DCT bidimensional de `A` se puede calcular como `B=D*A*D'`. Como `D` es una matriz real ortonormal, su inversa es igual a su traspuesta, luego la DCT inversa bidimensional de `B` será `D'*B*D`.

`n` es un entero escalar de tipo `double`. `D` es la matriz devuelta de tipo `double`.

Nota: Cuando A es cuadrada, a veces es más rápido usar la matriz DCT que la función `dct2`, especialmente si se está calculando un número elevado de pequeñas DCTs. Por ejemplo, en la compresión de imágenes JPEG se calcula la DCT de cada bloque de 8-por-8. Para ello se debe usar `dctmtx` para calcular la matriz D y posteriormente calcular cada DCT usando `D*A*D'` (donde A es cada bloque de 8-por-8). Con ello se evita calcular por separado la DCT de cada bloque de 8-por-8. Se puede ver un ejemplo en la función `blockproc`.

Ejemplos de uso

Ejemplo 1: Calcular la DCT a partir de la matriz de DCT.

```
% Leemos una imagen cuadrada en color y la convertimos a formato double.  
RGB = imread('autumn.tif');  
imshow(RGB)  
I = rgb2gray(RGB);  
A = im2double(I);  
  
% Sacamos la matriz DCT del tamaño de la imagen y calculamos la DCT.  
D = dctmtx(size(A,1));  
dct = D*A*D';  
figure, imshow(dct)
```

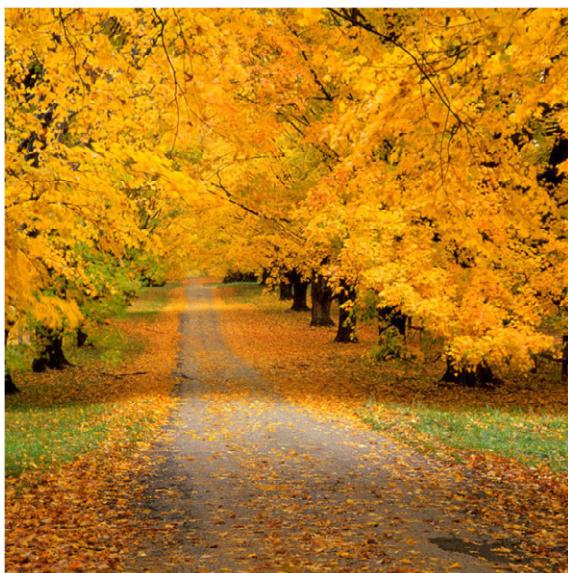


Figura f.107 - Imagen original (RGB)

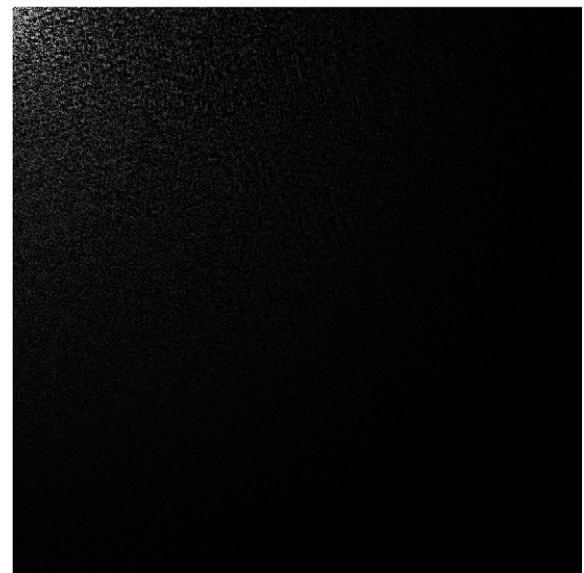


Figura f.109 - Energía de la imagen procesada J

Funciones relacionadas

`dct2`

deconvblind

Aclarado de una imagen borrosa (*deblurring*) usando deconvolución ciega

Introducción

La IPT incluye cuatro funciones para el aclarado de imágenes borrosas (*deblurring* o anti-desenfoque) listadas a continuación en orden de mayor a menor complejidad:

- `deconvwnr` - restaura la imagen utilizando el filtro Wiener.
- `deconvreg` - restaura la imagen utilizando un filtro regularizado.
- `deconvlucy` - restaura la imagen con el algoritmo Lucy-Richardson.
- `deconvblind` - restaura la imagen utilizando el algoritmo de deconvolución ciega.

El efecto borroso en una imagen puede deberse a muchos factores: movimiento durante la toma o captura de la imagen, debido a un largo periodo de exposición, óptica desenfocada, luz difusa, etc. Un modelo aproximado para describir a una imagen borrosa es el siguiente:
 $\mathbf{g} = \mathbf{H}\mathbf{f} + \mathbf{n}$, donde:

g = Imagen borrosa

H = Operador de distorsión o también llamado función de dispersión del punto (*point spread function* o PSF) que en el dominio del espacio describe el grado en el que un sistema óptico esparce o desenfoca un punto de luz. Este operador de distorsión, al convolucionarse con la imagen, crea la distorsión. Esta distorsión creada por el operador es solo uno de los distintos tipos de distorsión posibles.

f = La imagen original. Realmente no existe sino que representa lo que se hubiera obtenido en condiciones perfectas.

n Ruido aditivo, introducido durante la adquisición de la imagen y que corrompe la imagen.

Basándonos en este modelo, la tarea fundamental para aclarar una imagen borrosa consiste en deconvolucionar la imagen con el operador de dispersión PSF que describa exactamente su distorsión. La deconvolución es el proceso para revertir el efecto de la convolución. La calidad de la imagen aclarada depende principalmente del conocimiento del operador de

dispersión, cuya estimación apropiada es uno de los problemas más difíciles en la restauración de imágenes digitales.

Todas las funciones destinadas al aclarado de una imagen borrosa en la IPT aceptan como entrada un operador de dispersión PSF y una imagen borrosa a restaurar. Las funciones `deconvwnr` y `deconvreg` necesitan información a cerca del ruido para poder reducir la posible amplificación del ruido durante el proceso de enfoque o aclarado. Por otro lado, la función `deconvlucy` realiza múltiples iteraciones con técnicas de optimización y estadísticas de Poisson y no necesita tener información a cerca del ruido aditivo de la imagen degradada, al igual que la función `deconvblind`, que restaura la imagen sin el conocimiento del operador de dispersión (deconvolución ciega) basándose en una estimación inicial que se le pasa como parámetro e iterando de la misma manera que la función `deconvlucy`. En cualquiera de los casos siempre se tendrá que iterar varias veces y observar los resultados intermedios para encontrar la mejor aproximación a la imagen original.

Sintaxis

```
[J,PSF] = deconvblind(I, INITPSF)
[J,PSF] = deconvblind(I, INITPSF, NUMIT)
[J,PSF] = deconvblind(I, INITPSF, NUMIT, DAMPAR)
[J,PSF] = deconvblind(I, INITPSF, NUMIT, DAMPAR, WEIGHT)
[J,PSF] = deconvblind(I, INITPSF, NUMIT, DAMPAR, WEIGHT, READOUT)
[J,PSF] = deconvblind(..., FUN, P1, P2,...,PN)
```

Descripción

La función `deconvblind` realiza una deconvolución empezando donde acabó una deconvolución previa. Para ello se pasan la imagen de entrada `I` (imagen borrosa) y la estimación inicial del operador de dispersión PSF `INITPSF` como matrices de celdas `{I}` e `{INITPSF}`. `deconvblind` devuelve la imagen de salida `J` (imagen aclarada) y la restauración de PSF, de nuevo disponible para pasarse como array de entrada en la siguiente llamada a la función `deconvblind`.

`[J,PSF] = deconvblind(I, INITPSF)` deconvoluciona la imagen `I` usando el algoritmo de máxima probabilidad, devolviendo la imagen enfocada y el operador de dispersión `PSF` restaurado. Este operador `PSF` es un array positivo normalizado que tiene el mismo tamaño de que `INITPSF`. La restauración de `PSF` está más afectada por el tamaño de la estimación

inicial INITPSF que por sus valores y por ello se debe especificar INITPSF como un array de unos. I puede ser un array N-dimensional.

[J,PSF] = deconvblind(I, INITPSF, NUMIT) especifica el número de iteraciones (10 por defecto) en la variable NUMIT.

[J,PSF] = deconvblind(I, INITPSF, NUMIT, DAMPAR) especifica en la variable DAMPAR el umbral de desviación de la imagen resultante a partir de la imagen de entrada I (en términos de la desviación estándar del ruido de Poisson) por debajo del cual se produce amortiguación. Se suprimen las iteraciones para los píxeles que se desvían DAMPAR del valor original. Esto elimina la generación de ruido en esos píxeles y mantiene los detalles de la imagen en el resto. El valor por defecto es 0 (sin amortiguación).

[J,PSF] = deconvblind(I, INITPSF, NUMIT, DAMPAR, WEIGHT) especifica el peso que tendrán los píxeles de la imagen de entrada I en la restauración, reflejando la calidad de captación de la cámara. Si los píxeles tienen mala calidad, se les debe dar mucho peso para que se restauren fuertemente. Por defecto WEIGHT es un array unitario del mismo tamaño que la imagen de entrada y a sus elementos se les puede asignar un valor de entre 0.0 y 1 dependiendo si se quiere que sean menos o más considerados en la restauración. Por ejemplo, para excluir un pixel de ser considerado, se le debe asignar valor 0 en el array. Se puede ajustar el peso asignado a cada pixel en función de la cantidad de corrección de campo plano.

[J,PSF] = deconvblind(I, INITPSF, NUMIT, DAMPAR, WEIGHT, READOUT), donde READOUT es un array (o valor) correspondiente al ruido aditivo (p.ej. de fondo) y a la varianza del ruido de lectura (*Read out noise*) de la cámara. READOUT debe estar en las unidades de la imagen y su valor por defecto es 0.

[J,PSF] = deconvblind(..., FUN, P1, P2,...,PN), donde FUN es una referencia o puntero a una función que describe restricciones adicionales en el operador de dispersión PSF.

A FUN se le llama al final de cada iteración. FUN debe aceptar al operador de dispersión PSF como primer argumento y puede aceptar parámetros adicionales P1, P2,..., PN. La función

FUN debe devolver un argumento, PSF, que tenga el mismo tamaño que el PSF original y que cumpla las restricciones de positividad y normalización.

Nota: La imagen de salida \mathcal{J} podría tener *ringing* (artefactos oscilantes en imágenes restauradas) introducido por la Transformada de Fourier discreta usada en el algoritmo. Para reducirlo se debe usar $I = \text{edgetaper}(I, \text{PSF})$ antes de llamar a la función `deconvblind`.

La matriz de celdas de salida \mathcal{J} contiene cuatro elementos:

- $\mathcal{J}\{1\}$ contiene la imagen original I .
- $\mathcal{J}\{2\}$ contiene el resultado de la última iteración.
- $\mathcal{J}\{3\}$ contiene el resultado de la penúltima iteración.
- $\mathcal{J}\{4\}$ es un array generado por el algoritmo de iteración.

I e `INITPSF` pueden ser tipo `uint8`, `uint16`, `int16`, `single`, o `double`. `DAMPAR` y `READOUT` deben ser del mismo tipo que la imagen de entrada. El resto de variables de entrada deben ser de tipo `double`. La imagen de salida \mathcal{J} (o el primer array de la matriz de celdas de salida) tiene el mismo tipo que la imagen de entrada I . La variable de salida `PSF` es de tipo `double`.

Ejemplos de uso

Ejemplo 1: Aclarar una imagen borrosa debido a ruido gaussiano utilizando deconvolución ciega y comparar el resultado final e inicial tanto de la imagen como del elemento estructurante PSF.

```
% Creamos una imagen con patrón tablero de ajedrez.  
I = checkerboard(8);  
  
% Creamos un filtro gaussiano, nuestro operador de dispersión PSF.  
% Filtramos la imagen y le añadimos ruido.  
  
PSF = fspecial('gaussian', 7, 10);  
V = .0001;  
BlurredNoisy = imnoise(imfilter(I, PSF), 'gaussian', 0, V);  
  
% Definimos el peso de los píxeles.  
WT = zeros(size(I));  
WT(5:end-4, 5:end-4) = 1;
```

```
% Definimos el operador de distorsión inicial INITPSF como array de unos  
del tamaño de PSF.  
  
INITPSF = ones(size(PSF));  
  
% Realizamos la deconvolución ciega y comparamos la imagen original, la  
deconvolucionada/aclarada y los operadores de distorsión inicial y final.  
  
[J P] = deconvblind(BlurredNoisy,INITPSF,20,10*sqrt(V),WT);  
subplot(221);imshow(BlurredNoisy)  
title('Imagen original Ruidosa y Borrosa');  
subplot(222);imshow(PSF,[])  
title('Coeficiente PSF verdadero');  
subplot(223);imshow(J)  
title('Imagen final aclarada');  
subplot(224);imshow(P,[])  
title('Coeficiente PSF recuperado');
```

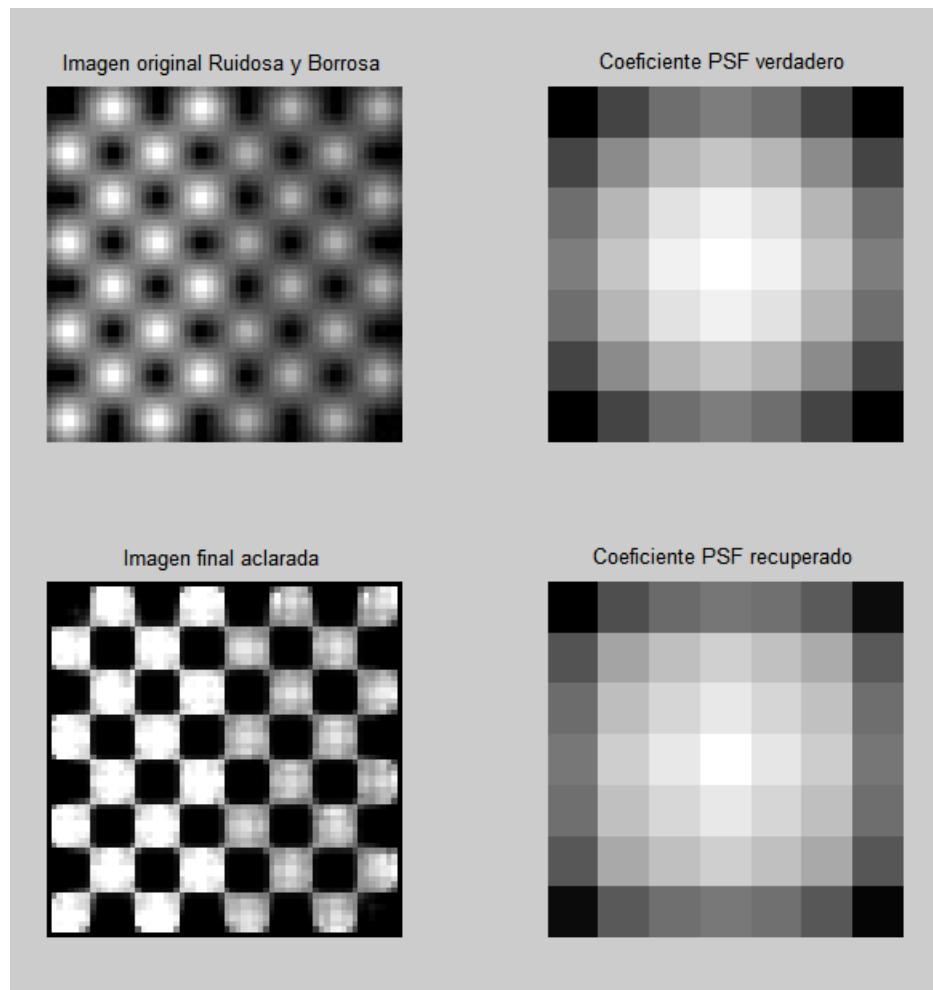


Figura f.110 - comparativa deconvblind

Funciones relacionadas

`deconvlucy`, `deconvreg`, `deconvwnr`, `edgetaper`, `function_handle`, `imnoise`, `otf2psf`, `padarray`, `psf2otf`

deconvlucy

Aclarado de una imagen usando el método Lucy-Richardson

Introducción

La IPT incluye cuatro funciones para el aclarado de imágenes borrosas (*deblurring* o anti-desenfoque) listadas a continuación en orden de mayor a menor complejidad:

- `deconvwnr` - restaura la imagen utilizando el filtro Wiener.
- `deconvreg` - restaura la imagen utilizando un filtro regularizado.
- `deconvlucy` - restaura la imagen con el algoritmo Lucy-Richardson.
- `deconvblind` - restaura la imagen utilizando el algoritmo de deconvolución ciega.

La tarea fundamental para aclarar una imagen borrosa consiste en deconvolucionar la imagen con el operador de dispersión PSF que describa exactamente su distorsión (más detalle en la función `deconvblind`). La deconvolución es el proceso para revertir el efecto de la convolución. La calidad de la imagen aclarada depende principalmente del conocimiento del operador de dispersión, cuya estimación apropiada es uno de los problemas más difíciles en la restauración de imágenes digitales.

Todas las funciones destinadas al aclarado de una imagen borrosa en la IPT aceptan como entrada un operador de dispersión PSF y una imagen borrosa a restaurar. Las funciones `deconvwnr` y `deconvreg` necesitan información a cerca del ruido para poder reducir la posible amplificación del ruido durante el proceso de enfoque o aclarado. Por otro lado, la función `deconvlucy` realiza múltiples iteraciones con técnicas de optimización y estadísticas de Poisson y no necesita tener información a cerca del ruido aditivo de la imagen degradada, al igual que la función `deconvblind`, que restaura la imagen sin el conocimiento del operador de dispersión (deconvolución ciega) basándose en una estimación inicial que se le pasa como parámetro e iterando de la misma manera que la función `deconvlucy`. En cualquiera de los casos siempre se tendrá que iterar varias veces y observar los resultados intermedios para encontrar la mejor aproximación a la imagen original.

Sintaxis

```
J = deconvlucy(I, PSF)
J = deconvlucy(I, PSF, NUMIT)
J = deconvlucy(I, PSF, NUMIT, DAMPAR)
J = deconvlucy(I, PSF, NUMIT, DAMPAR, WEIGHT)
J = deconvlucy(I, PSF, NUMIT, DAMPAR, WEIGHT, READOUT)
J = deconvlucy(I, PSF, NUMIT, DAMPAR, WEIGHT, READOUT, SUBSMPL)
```

Descripción

`J = deconvlucy(I, PSF)` arregla una imagen `I` que ha sido degradada debido a una convolución con un operador de dispersión `PSF` y probablemente también a la adición de ruido aditivo. `I` puede ser un array N-dimensional.

`J = deconvlucy(I, PSF, NUMIT)` especifica el número de iteraciones (10 por defecto) en la variable `NUMIT`.

`J = deconvlucy(I, PSF, NUMIT, DAMPAR)` especifica en la variable `DAMPAR` el umbral de desviación de la imagen resultante a partir de la imagen de entrada `I` (en términos de la desviación estándar del ruido de Poisson) por debajo del cual se produce amortiguación. Se suprimen las iteraciones para los píxeles que se desvían `DAMPAR` del valor original. Esto elimina la generación de ruido en esos píxeles y mantiene los detalles de la imagen en el resto. El valor por defecto es 0 (sin amortiguación).

`J = deconvlucy(I, PSF, NUMIT, DAMPAR, WEIGHT)` especifica el peso que tendrán los píxeles de la imagen de entrada `I` en la restauración, reflejando la calidad de captación de la cámara. Si los píxeles tienen mala calidad, se les debe dar mucho peso para que se restauren fuertemente. Por defecto `WEIGHT` es un array unitario del mismo tamaño que la imagen de entrada y a sus elementos se les puede asignar un valor de entre 0.0 y 1 dependiendo si se quiere que sean menos o más considerados en la restauración. Por ejemplo, para excluir un pixel se le debe asignar valor 0 en el array.

`J = deconvlucy(I, PSF, NUMIT, DAMPAR, WEIGHT, READOUT)` donde `READOUT` es un array (o valor) correspondiente al ruido aditivo (p.ej. de fondo) y la varianza del ruido de lectura de la cámara. `READOUT` debe estar en las mismas unidades que la imagen y su valor por defecto es 0.

`J = deconvlucy(I, PSF, NUMIT, DAMPAR, WEIGHT, READOUT, SUBSMPL)`, donde la variable `SUBSMPL` significa sub-muestreo y se usa cuando el coeficiente `PSF` se da en una cuadrícula que es `SUBSMPL` veces más fina que la imagen. El valor por defecto es 1.

Nota: La imagen de salida `J` podría tener *ringing* (artefactos oscilantes en imágenes restauradas) introducido por la Transformada de Fourier discreta usada en el algoritmo. Para reducirlo se debe usar `I = edgetaper(I,PSF)` antes de llamar a la función `deconvlucy`.

Si `I` es una matriz de celdas, esta puede contener un array numérico único (la imagen borrosa) o puede ser la salida de una llamada anterior a la función `deconvlucy`. Cuando se le pasa una matriz de celdas como entrada, devuelve un array de celdas `J` de 1-por-4 en el que:

- `J{1}` contiene la imagen original `I`.
- `J{2}` contiene el resultado de la última iteración.
- `J{3}` contiene el resultado de la penúltima iteración.
- `J{4}` es un array generado por el algoritmo de iteración.

`I` y `PSF` pueden ser tipo `uint8`, `uint16`, `int16`, `single`, o `double`. `DAMPAR` y `READOUT` deben ser del mismo tipo que la imagen de entrada. El resto de variables de entrada deben ser de tipo `double`. La imagen de salida `J` (o el primer array de la matriz de celdas de salida) tiene el mismo tipo que la imagen de entrada `I`.

Ejemplos de uso

Ejemplo 1: Aclarar una imagen borrosa debido a ruido gaussiano utilizando el método Lucy-Richardson y comparar el resultado final e inicial utilizando varios parámetros de la función.

```
% Creamos una imagen con patrón tablero de ajedrez.  
I = checkerboard(8);  
  
% Creamos un filtro gaussiano, nuestro operador de dispersión PSF.  
% Filtramos la imagen y le añadimos ruido.  
  
PSF = fspecial('gaussian',7,10);  
V = .0001;  
BlurredNoisy = imnoise(imfilter(I,PSF), 'gaussian', 0, V);
```

% Definimos el peso de los píxeles.

```
WT = zeros(size(I));  
WT(5:end-4,5:end-4) = 1;
```

% Realizamos la deconvolución ciega y comparamos la imagen original con los resultados de la deconvolución utilizando varios parámetros (iteraciones, umbral de desviación y peso de los píxeles).

```
J1 = deconvlucy(BlurredNoisy,PSF);  
J2 = deconvlucy(BlurredNoisy,PSF,20,sqrt(V));  
J3 = deconvlucy(BlurredNoisy,PSF,20,sqrt(V),WT);  
  
subplot(221);imshow(BlurredNoisy)  
title('Imagen original Ruidosa y Borrosa');  
subplot(222);imshow(J1)  
title('deconvlucy(A,PSF)');  
subplot(223);imshow(J2)  
title('deconvlucy(A,PSF,NI,DP)');  
subplot(224);imshow(J3)  
title('deconvlucy(A,PSF,NI,DP,WT)');
```

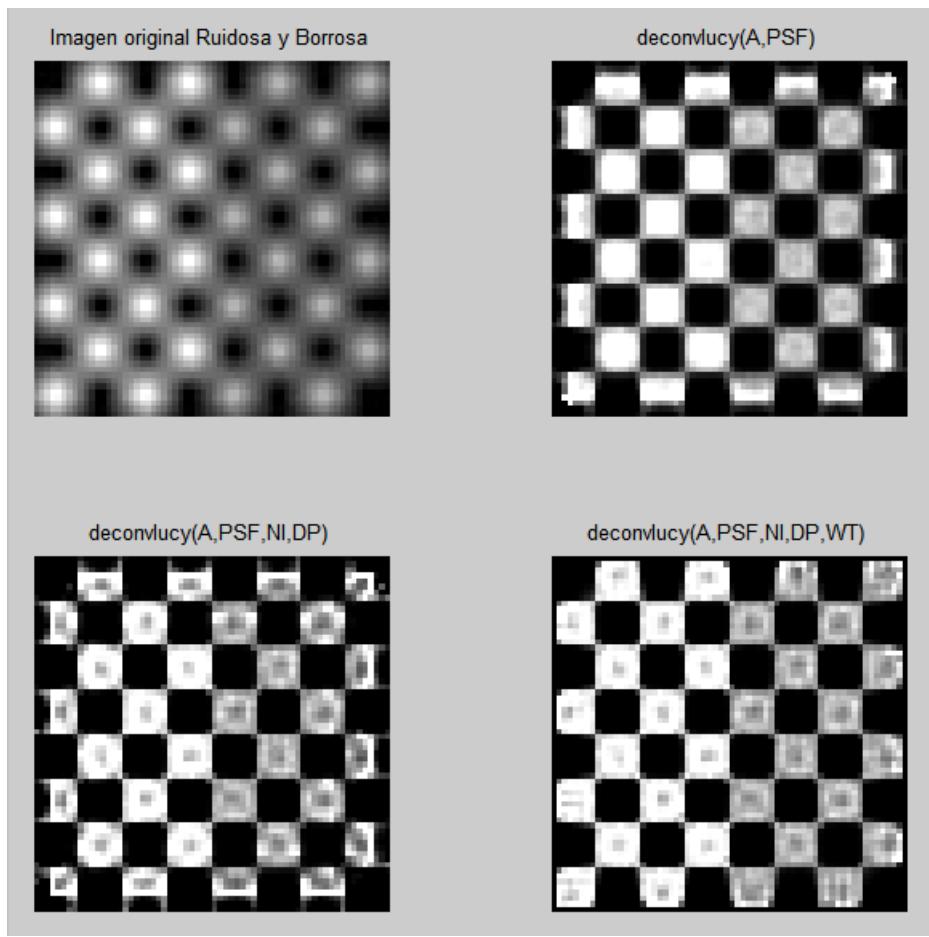


Figura f.111 - comparativa deconvlucy

Funciones relacionadas

`deconvblind`, `deconvreg`, `deconvwnr`, `otf2psf`, `padarray`, `psf2otf`

deconvreg

Aclarado de una imagen usando el método del filtro normalizado

Introducción

La IPT incluye cuatro funciones para el aclarado de imágenes borrosas (*deblurring* o anti-desenfoque) listadas a continuación en orden de mayor a menor complejidad:

- `deconvwnr` - restaura la imagen utilizando el filtro Wiener.
- `deconvreg` - restaura la imagen utilizando un filtro regularizado.
- `deconvlucy` - restaura la imagen con el algoritmo Lucy-Richardson.
- `deconvblind` - restaura la imagen utilizando el algoritmo de deconvolución ciega.

La tarea fundamental para aclarar una imagen borrosa consiste en deconvolucionar la imagen con el operador de dispersión PSF que describa exactamente su distorsión (más detalle en la función `deconvblind`). La deconvolución es el proceso para revertir el efecto de la convolución. La calidad de la imagen aclarada depende principalmente del conocimiento del operador de dispersión, cuya estimación apropiada es uno de los problemas más difíciles en la restauración de imágenes digitales.

Todas las funciones destinadas al aclarado de una imagen borrosa en la IPT aceptan como entrada un operador de dispersión PSF y una imagen borrosa a restaurar. Las funciones `deconvwnr` y `deconvreg` necesitan información a cerca del ruido para poder reducir la posible amplificación del ruido durante el proceso de enfoque o aclarado. Por otro lado, la función `deconvlucy` realiza múltiples iteraciones con técnicas de optimización y estadísticas de Poisson y no necesita tener información a cerca del ruido aditivo de la imagen degradada, al igual que la función `deconvblind`, que restaura la imagen sin el conocimiento del operador de dispersión (deconvolución ciega) basándose en una estimación inicial que se le pasa como parámetro e iterando de la misma manera que la función `deconvlucy`. En cualquiera de los casos siempre se tendrá que iterar varias veces y observar los resultados intermedios para encontrar la mejor aproximación a la imagen original.

Sintaxis

```
J = deconvreg(I, PSF)
J = deconvreg(I, PSF, NOISEPOWER)
J = deconvreg(I, PSF, NOISEPOWER, LRANGE)
J = deconvreg(I, PSF, NOISEPOWER, LRANGE, REGOP)
[J, LAGRA] = deconvreg(I, PSF,...)
```

Descripción

`J = deconvreg(I, PSF)` deconvoluciona la imagen `I` utilizando el algoritmo del filtro normalizado. La función devuelve la imagen aclarada `J` asumiendo que la imagen `I` se creó a partir de la convolución de la imagen con un operador de dispersión `PSF` y probablemente con la adición de ruido. El algoritmo está limitado de forma óptima ya que hay el menor error cuadrado posible entre las imágenes estimadas y reales, preservando la suavidad de la imagen. `I` puede ser un array N-dimensional.

`J = deconvreg(I, PSF, NOISEPOWER)` donde `NOISEPOWER` es la fuerza del ruido aditivo. Por defecto es 0.

`J = deconvreg(I, PSF, NOISEPOWER, LRANGE)` donde `LRANGE` es un vector que especifica el rango en el que se buscará la solución óptima. El algoritmo encontrará un multiplicador Lagrange óptimo `LAGRA` dentro del rango `LRANGE`. Si `LRANGE` es un escalar, el algoritmo asume que se ha dado `LAGRA` y que vale lo mismo que `LRANGE`; el valor `NP` se ignoraría en ese caso. El rango por defecto está comprendido entre [1e-9 y 1e9].

`J = deconvreg(I, PSF, NOISEPOWER, LRANGE, REGOP)` donde `REGOP` es el operador de regularización que hará la deconvolución. Por defecto es el operador Laplaciano, para mantener la suavidad de la imagen. Las dimensiones del array `REGOP` no deben exceder las dimensiones de la imagen; cualquier dimensión no única debe corresponderse con la dimensión no única de `PSF`.

`[J, LAGRA] = deconvreg(I, PSF,...)` guarda el valor del multiplicador de Lagrange en la variable `LAGRA` además de restaurar la imagen `J`.

Nota: La imagen de salida `J` podría tener *ringing* (artefactos oscilantes en imágenes restauradas) introducido por la Transformada de Fourier discreta usada en

el algoritmo. Para reducirlo se debe usar `I = edgetaper(I,PSF)` antes de llamar a la función `deconvreg`.

`I` puede ser de tipo `uint8`, `uint16`, `int16`, `single`, o `double`. El resto de parámetros de entrada deberán ser de tipo `double`. `J` será de la misma clase que `I`.

Ejemplos de uso

Ejemplo 1: Aclarar una imagen borrosa debido a ruido gaussiano utilizando el método del filtro normalizado y comparar el resultado final e inicial utilizando varios parámetros de la función.

```
% Creamos una imagen con patrón tablero de ajedrez.  
I = checkerboard(8);  
  
% Creamos un filtro gaussiano, nuestro operador de dispersión PSF.  
% Filtramos la imagen y le añadimos ruido.  
  
PSF = fspecial('gaussian',7,10);  
V = .01;  
BlurredNoisy = imnoise(imfilter(I,PSF),'gaussian',0,V);  
  
NOISEPOWER = V*prod(size(I));  
  
% Realizamos la deconvolución y recogemos la imagen deconvolucionada y el  
% multiplicador de Lagrange.  
  
[J LAGRA] = deconvreg(BlurredNoisy,PSF,NOISEPOWER);  
  
% Comparamos la imagen original con los resultados de la deconvolución  
% utilizando varios parámetros (fuerza del ruido aditivo y rango Lagrange  
% disminuido y aumentado).  
  
subplot(221); imshow(BlurredNoisy)  
title('Imagen original Ruidosa y Borrosa');  
subplot(222); imshow(J)  
title('[J LAGRA] = deconvreg(A,PSF,NP)');  
subplot(223); imshow(deconvreg(BlurredNoisy,PSF,[],LAGRA/10))  
title('deconvreg(A,PSF,[],0.1*LAGRA)');  
subplot(224); imshow(deconvreg(BlurredNoisy,PSF,[],LAGRA*10))  
title('deconvreg(A,PSF,[],10*LAGRA)' );
```

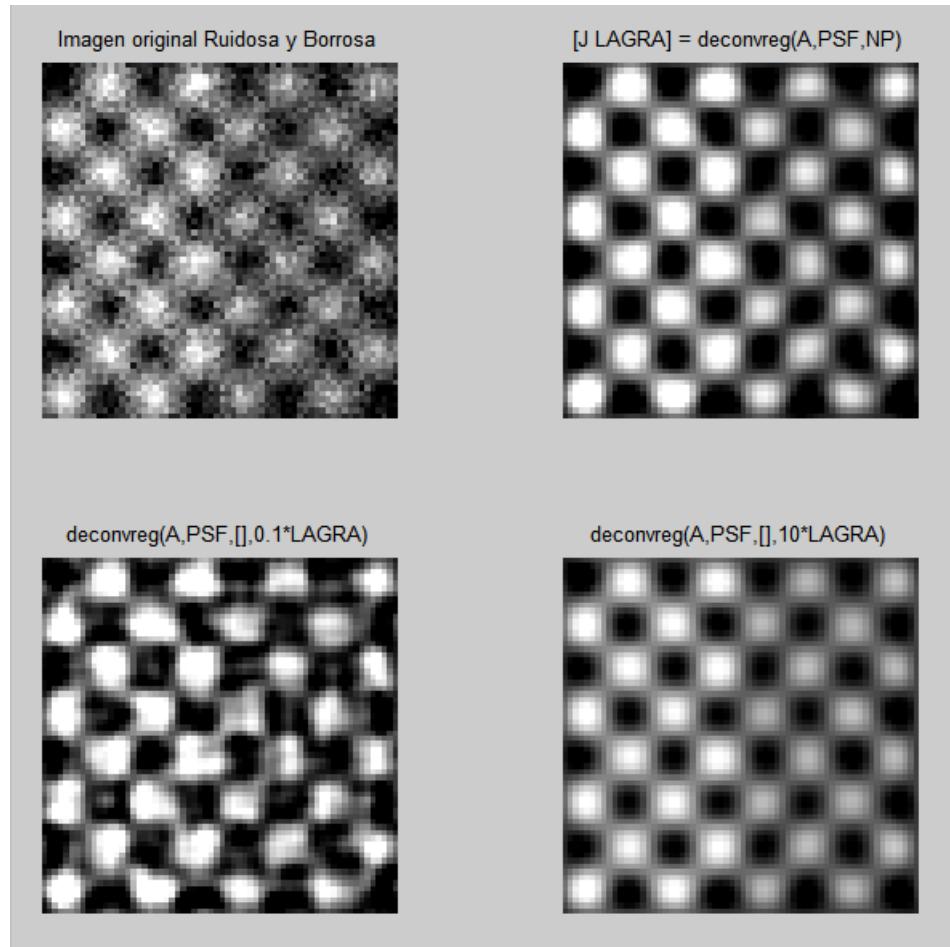


Figura f.112 - comparativa deconvreg

Funciones relacionadas

deconvblind, deconvlucy, deconvwnr, otf2psf, padarray, psf2otf

deconvwnr

Aclarado de una imagen usando un filtro Wiener

Introducción

La IPT incluye cuatro funciones para el aclarado de imágenes borrosas (*deblurring* o anti-desenfoque) listadas a continuación en orden de mayor a menor complejidad:

- `deconvwnr` - restaura la imagen utilizando el filtro Wiener.
- `deconvreg` - restaura la imagen utilizando un filtro regularizado.
- `deconvlucy` - restaura la imagen con el algoritmo Lucy-Richardson.
- `deconvblind` - restaura la imagen utilizando el algoritmo de deconvolución ciega.

La tarea fundamental para aclarar una imagen borrosa consiste en deconvolucionar la imagen con el operador de dispersión PSF que describa exactamente su distorsión (más detalle en la función `deconvblind`). La deconvolución es el proceso para revertir el efecto de la convolución. La calidad de la imagen aclarada depende principalmente del conocimiento del operador de dispersión, cuya estimación apropiada es uno de los problemas más difíciles en la restauración de imágenes digitales.

Todas las funciones destinadas al aclarado de una imagen borrosa en la IPT aceptan como entrada un operador de dispersión PSF y una imagen borrosa a restaurar. Las funciones `deconvwnr` y `deconvreg` necesitan información a cerca del ruido para poder reducir la posible amplificación del ruido durante el proceso de enfoque o aclarado. Por otro lado, la función `deconvlucy` realiza múltiples iteraciones con técnicas de optimización y estadísticas de Poisson y no necesita tener información a cerca del ruido aditivo de la imagen degradada, al igual que la función `deconvblind`, que restaura la imagen sin el conocimiento del operador de dispersión (deconvolución ciega) basándose en una estimación inicial que se le pasa como parámetro e iterando de la misma manera que la función `deconvlucy`. En cualquiera de los casos siempre se tendrá que iterar varias veces y observar los resultados intermedios para encontrar la mejor aproximación a la imagen original.

Sintaxis

```
J = deconvwnr(I,PSF,NSR)
J = deconvwnr(I,PSF,NCORR,ICORR)
```

Descripción

`J = deconvwnr(I,PSF,NSR)` deconvoluciona la imagen `I` utilizando el filtro de Wiener como algoritmo y devolviendo la imagen aclarada `J`. `I` puede ser un array N-dimensional. `PSF` es el operador de dispersión con el que se convolucionó la imagen `I`. `NSR` es el coeficiente de señal a ruido del ruido aditivo y puede ser un escalar o un array del mismo tamaño que `I`. Dar a `NSR` un valor de 0 es equivalente a crear un filtro inverso ideal.

`J = deconvwnr(I,PSF,NCORR,ICORR)` deconvoluciona la imagen `I`, con `NCORR` representando la función de autocorrelación del ruido e `ICORR` la función de autocorrelación de la imagen original. `NCORR` e `ICORR` pueden ser de cualquier dimensión, no excediendo la de la imagen original. Si `NCORR` o `ICORR` son arrays N-dimensionales, sus valores corresponden con la autocorrelación en cada dimensión. Si `NCORR` o `ICORR` son vectores y `PSF` también, los valores representan la función de autocorrelación en la primera dimensión. Si `PSF` es un array, la función de autocorrelación de la primera dimensión se extrae por simetría a todas las dimensiones no únicas de `PSF`. Si `NCORR` o `ICORR` son un escalar, su valor representa la potencia de ruido de la imagen.

Nota: La imagen de salida `J` podría tener *ringing* (artefactos oscilantes en imágenes restauradas) introducido por la Transformada de Fourier discreta usada en el algoritmo. Para reducirlo se debe usar `I = edgetaper(I,PSF)` antes de llamar a la función `deconvwnr`.

`I` puede ser del tipo `uint8`, `uint16`, `int16`, `single`, o `double`. El resto de entradas deben ser de clase `double`. `J` será de la misma clase que `I`.

Ejemplos de uso

Ejemplo 1: Aclarar una imagen borrosa debido a desenfoque por movimiento y a ruido gaussiano utilizando el filtro Wiener y comparar el resultado final e inicial. Utilizar el filtro directamente suponiendo que no hay ruido y también estimando el posible ruido.

```
% Mostramos la imagen original.

I = im2double(imread('cameraman.tif'));
subplot(221); imshow(I)
title('Imagen Original');

% Simulamos desenfoque por movimiento.

LEN = 21;
THETA = 11;
PSF = fspecial('motion', LEN, THETA);
blurred = imfilter(I, PSF, 'conv', 'circular');

% Simulamos ruido aditivo.

noise_mean = 0;
noise_var = 0.0001;
blurred_noisy = imnoise(blurred, 'gaussian', noise_mean, noise_var);
subplot(222); imshow(blurred_noisy)
title('Simular Ruido y Desenfoque');

% Intentamos restaurarla suponiendo que no hay ruido.

estimated_nsr = 0;
wnr2 = deconvwnr(blurred_noisy, PSF, estimated_nsr);
subplot(223); imshow(wnr2)
title('Restauración usando NSR = 0');

% Intentamos restaurarla utilizando una mejor estimación del ratio de
señal a ruido.

estimated_nsr = noise_var / var(I(:));
wnr3 = deconvwnr(blurred_noisy, PSF, estimated_nsr);
subplot(224); imshow(wnr3)
title('Restauración usando una estimación de NSR');
```

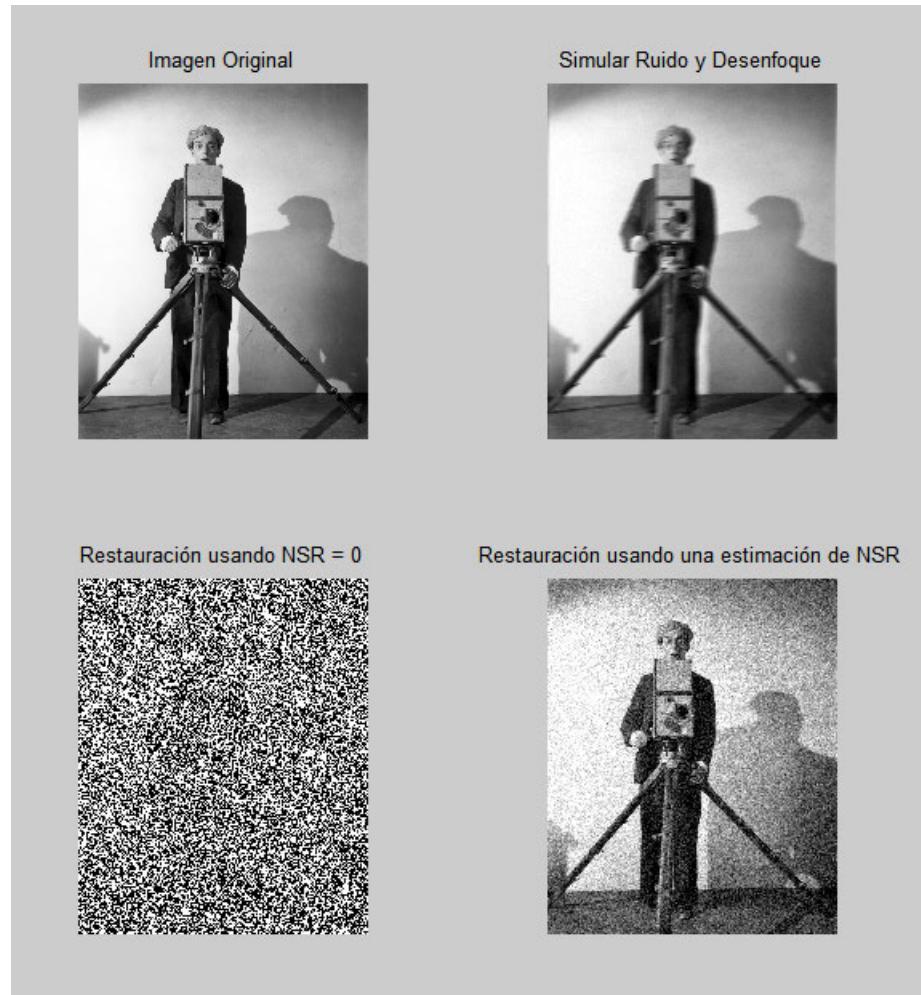


Figura f.113 - comparativa deconvwnr

Funciones relacionadas

`deconvblind`, `deconvlucy`, `deconvreg`, `edgetaper`, `otf2psf`, `padarray`, `psf2otf`

decorrstretch

Descorrelación (*decorrelation stretch*) en una imagen multicanal

Introducción

Decorrelation stretch (o simplemente descorrelación) es una técnica de mejora de imágenes que saca el máximo partido a las diferencias de color en una imagen. Se empezó aplicando en teledetección y tiene la habilidad de destacar elementos de la imagen que son prácticamente invisibles al ojo humano. Una de las formas de hacerlo es eliminando la correlación intercanal de los píxeles de entrada, de ahí el nombre de *decorrelation stretch*.

Sintaxis

```
S = decorrstretch(I)
S = decorrstretch(I, 'TOL', TOL)
```

Descripción

`S = decorrstretch(I)` aplica la técnica de *decorrelation stretch* a una imagen multicanal `I` y devuelve el resultado en la variable `S` del mismo tamaño y clase que `I`. La media y la varianza en cada canal o banda son las mismas que en `I`.

`S = decorrstretch(I, 'TOL', TOL)` mejora el contraste después de aplicar la técnica de *decorrelation stretch*. La mejora del contraste se controla con la variable `TOL`. `TOL` debe estar entre 0 y 0.5.

- `TOL = [LOW_FRACT HIGH_FRACT]` especifica la fracción de la imagen que se saturará usando límites de intensidad alta y baja.
- Si `TOL` es un escalar, `LOW_FRACT = TOL`, y `HIGH_FRACT = 1 - TOL`, lo que hace que se saturen fracciones de imagen iguales en baja y altas intensidades.

Nota: La técnica de *decorrelation stretch* se aplica normalmente a imágenes multicanal de 3 canales (normalmente imágenes RGB) pero `decorrstretch` trabaja con un número arbitrario de canales o bandas.

El principal propósito de esta técnica es la de mejorar la visualización de una imagen. Pequeños ajustes en la variable `TOL` pueden significar grandes cambios en la apariencia de la salida.

La imagen de entrada debe de ser de tipo `uint8`, `uint16`, `int16`, `single`, o `double`.

Ejemplos de uso

Ejemplo 1: Destacar elementos importantes en unas imágenes comparando el resultado.

```
% Leemos la imagen y le aplicamos la descorrelación usando distintos
valores de contraste.

I = imread('earth.tif');

S = decorrstretch(I,'TOL',0.01);
S2 = decorrstretch(I,'TOL',0.1);
TOL = [0.01 0.2];
S3 = decorrstretch(I,'TOL',TOL);

% Mostramos todas las imágenes.

figure, imshow(I)
figure, imshow(S)
figure, imshow(S2)
figure, imshow(S3)

% Leemos otra imagen, le aplicamos de nuevo la descorrelación y
mostramos.

IS = imread('altamira.tif');

SS = decorrstretch(IS,'TOL',0.01);

figure, imshow(IS)
figure, imshow(SS)
```



Figura f.114 - Imagen original (I)

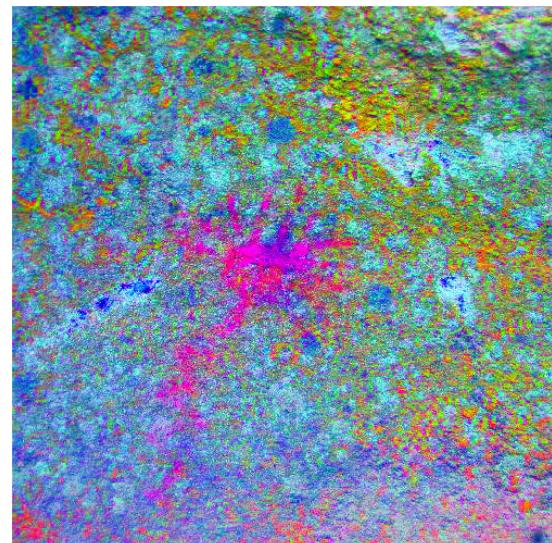


Figura f.115 - Imagen procesada (S)

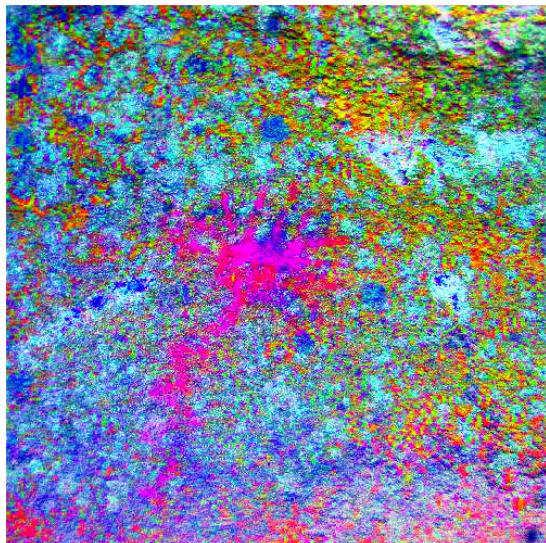


Figura f.116 - Imagen procesada (S2)

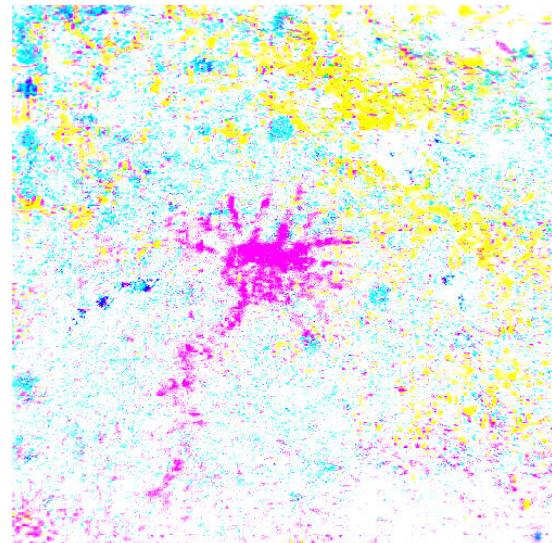


Figura f.117 - Imagen procesada (S3)



Figura f.118 - Imagen original (IS)

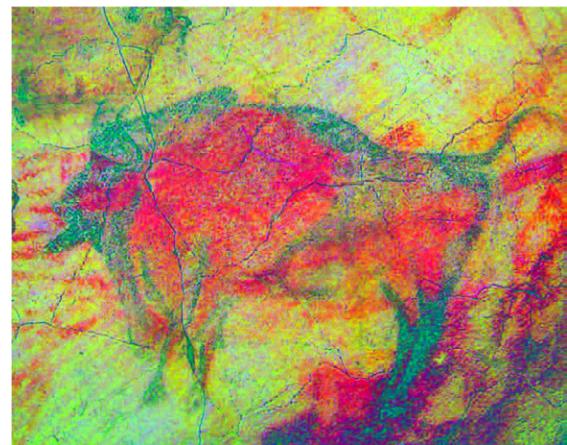


Figura f.119 - Imagen procesada (SS)

Funciones relacionadas

`imadjust`, `stretchlim`

demosaic

Conversión de una imagen en formato patrón Bayer a color verdadero

Introducción

El filtro, máscara o mosaico de Bayer (o filtro de color) es un tipo de matriz de filtros rojos verdes y azules, que se sitúa sobre un sensor digital de imagen para hacer llegar a cada fotodiodo la información de luminosidad correspondiente a una sección de los distintos colores primarios. Interpolando las muestras de cuatro fotodiodes vecinos se obtiene un pixel de color. El mosaico de Bayer está formado por un 50% de filtros verdes, un 25% de rojos y un 25% de azules. Interpolando dos muestras verdes, una roja y una azul se obtiene un pixel de color. La razón de que se use mayor cantidad de puntos verdes es que el ojo humano es más sensible a ese color. ^[11]

Sintaxis

```
RGB = demosaic(I, sensorAlignment)
```

Descripción

RGB = demosaic(I, sensorAlignment) convierte una imagen codificada con un patrón Bayer a una imagen en formato color verdadero utilizando interpolación lineal. I es un array de M-por-N con valores de intensidad que están codificados en formato Bayer. I debe tener al menos 5 filas y 5 columnas.

La función `demosaic` utiliza interpolación para convertir la imagen bidimensional codificada con el patrón Bayer en una imagen en formato color verdadero que es un array de M-por-N-por-3.

`sensorAlignment` es una de las siguientes cadenas de texto que define el patrón Bayer. Cada cadena representa el orden de los sensores rojos, verdes y azules describiendo los cuatro píxeles en la esquina superior izquierda de la imagen (de izquierda a derecha y de arriba a abajo):

Valor cadena	Alineación 2-por-2 del Sensor
'gbrg'	
'grbg'	
'bggr'	
'rggb'	

I puede ser uint8 o uint16, y debe ser real. *RGB* es del mismo tipo que *I*.

Ejemplos de uso

Ejemplo 1: Convertir una imagen codificada con un patrón Bayer que ha sido tomada por una cámara de fotos con una alineación del sensor del tipo 'bggr'.

% Mostramos la imagen codificada en patrón Bayer, convertimos a color verdadero especificando la alineación del sensor y mostramos resultado.

```
I = imread('mandi.tif');
J = demosaic(I,'bggr');
imshow(I)
figure, imshow(J)
```



Figura f.124 - Imagen original (I) patrón Bayer



Figura f.125 - Imagen procesada (J) convertida

dicomanon

Anonimizar un archivo DICOM

Introducción

DICOM (*Digital Imaging and Communication in Medicine*) es el estándar en Imagen Digital y Comunicaciones en Medicina desarrollado para que usuarios y fabricantes de equipamiento de imagen médica (tal como TAC, resonancia magnética, medicina nuclear y ultrasonidos) puedan intercambiar imágenes con otros dispositivos. El Colegio Estadounidense de Radiología, ACR y la Asociación Nacional de Fabricantes Eléctricos, NEMA llegaron al formato DICOM desarrollando una interfaz que permitía la comunicación entre el equipo y cualquier otro dispositivo y un diccionario de los elementos de datos necesarios para la visualización e interpretación de las imágenes.^[12]

Sintaxis

```
dicomanon(file_in, file_out)
dicomanon(..., 'keep', FIELDS)
dicomanon(..., 'update', ATTRS)
```

Descripción

`dicomanon(file_in, file_out)` elimina información médica confidencial del archivo DICOM especificado en `file_in` y crea un nuevo archivo `file_out` con los valores modificados. El resto de información no se modifica.

`dicomanon(..., 'keep', FIELDS)` modifica todos los datos confidenciales a excepción de los que están listados en el array `FIELDS`, que es un array de celdas con nombres de campo. Esta sintaxis es útil para mantener los metadatos que no identifican al paciente de forma única pero que son útiles para el diagnóstico (p.ej., *PatientAge* (edad del paciente), *PatientSex* (sexo del paciente), etc.).

Nota: Mantener ciertos campos puede comprometer la confidencialidad del paciente.

`dicomanon(..., 'update', ATTRS)` modifica los datos confidenciales manteniendo ciertos campos y actualizando otros campos confidenciales particulares. `ATTRS` es la

estructura cuyos campos son los nombres de los atributos a conservar. Usar esta sintaxis para mantener la jerarquía Estudios/Series/Imágenes o para reemplazar un valor específico con una propiedad más genérica (p.ej., eliminar la fecha de nacimiento del paciente (*PatientBirthDate*) manteniendo su edad (*PatientAge*)).

Para más información sobre los campos que se modificarán y eliminarán, consultar el suplemento DICOM 55 desde su web oficial <http://medical.nema.org/>.

Ejemplos de uso

Ejemplo 1: Eliminar todos los metadatos confidenciales de un archivo DICOM.

```
dicomanon('patient.dcm', 'anonymized.dcm')
```

Ejemplo 2: Eliminar todos los datos confidenciales de un archivo DICOM a excepción de la edad, el sexo y la descripción del estudio.

```
dicomanon('tumor.dcm', 'tumor_anon.dcm', 'keep', {'PatientAge', 'PatientSex', 'StudyDescripción'})
```

Ejemplo 3: Anonimizar una serie de imágenes manteniendo la jerarquía.

```
Valors.StudyInstanceUID = dicomuid;
Valors.SeriesInstanceUID = dicomuid;

d = dir('*.*');
for p = 1:numel(d)
    dicomanon(d(p).name, sprintf('anon%d.dcm', p), ...
        'update', Valors)
end
```

Funciones relacionadas

`dicominfo`, `dicomwrite`

dicomdict

Conseguir o activar los datos del diccionario DICOM

Introducción

El diccionario DICOM es el diccionario de los elementos de datos necesarios para la visualización e interpretación de las imágenes en el formato DICOM.

Sintaxis

```
dicomdict('set',dictionary)
dictionary = dicomdict('get')
dicomdict('factory')
```

Descripción

`dicomdict('set',dictionary)` asigna el diccionario DICOM al valor guardado en la variable `dictionary`, una cadena que contiene el nombre de archivo del diccionario. Las funciones utilizadas en el formato DICOM usan este diccionario por defecto a menos que se especifique otro diccionario desde la línea de comandos.

`dictionary = dicomdict('get')` devuelve una cadena que contiene el nombre de archivo del diccionario de los elementos de datos DICOM.

`dicomdict('factory')` resetea el diccionario DICOM a su valor inicial por defecto.

Nota: El diccionario es por defecto un archivo de MATLAB llamado `dicomdict.mat`. La *toolbox* incluye además una versión en texto del diccionario llamada `dicom-dict.txt`. Si se quiere crear un diccionario diferente, abrir el archivo `dicom-dict.txt` en un editor de texto, modificarlo y guardarla con otro nombre.

Ejemplos de uso

Ejemplo 1: Recoger el nombre del archivo de datos del diccionario DICOM.

```
dictionary = dicomdict('get')
```

Funciones relacionadas

`dicominfo`, `dicomread`, `dicomwrite`

dicominfo

Lectura de metadatos de un archivo DICOM

Sintaxis

```
info = dicominfo(filename)
info = dicominfo(filename, 'dictionary', D)
```

Descripción

info = dicominfo(filename) lee los metadatos del archivo en formato DICOM especificado en la cadena filename y lo guarda como estructura de metadatos en la variable info.

info = dicominfo(filename, 'dictionary', D) usa el archivo de diccionario de datos especificado en la variable D para leer el archivo en formato DICOM. El archivo especificado en D debe estar en la ruta de búsqueda de MATLAB. El archivo de diccionario por defecto es dicom-dict.mat.

Ejemplos de uso

Ejemplo 1: Leer los metadatos de un archivo en formato DICOM.

```
info = dicominfo('CT-MONO2-16-ankle.dcm')

info =
    Filename: [1x75 char]
    FileModDate: '18-dic-2000 12:06:42'
    FileSize: 525436
    Format: 'DICOM'
    FormatVersion: 3
    Width: 512
    Height: 512
    BitDepth: 16
    ColorType: 'grayscale'
    FileMetaInformationGroupLength: 192
    FileMetaInformationVersion: [2x1 uint8]
    MediaStorageSOPClassUID: '1.2.840.10008.5.1.4.1.1.7'
    MediaStorageSOPInstanceUID:
    '1.2.840.113619.2.1.2411.1031152382.365.1.736169244'
    TransferSyntaxUID: '1.2.840.10008.1.2'
    ImplementationClassUID: '1.2.840.113619.6.5'
    ImplementationVersionName: '1_2_5'
    SourceApplicationEntityTitle: 'CTN_STORAGE'
    IdentifyingGroupLength: 414
    ImageType: 'DERIVED\SECONDARY\3D'
    SOPClassUID: '1.2.840.10008.5.1.4.1.1.7'
```

```
SOPInstanceUID:  
'1.2.840.113619.2.1.2411.1031152382.365.1.736169244'  
    StudyDate: '1993.04.30'  
    SeriesDate: '1993.04.30'  
    ContentDate: '1993.04.30'  
    StudyTime: '11:27:24'  
    SeriesTime: '11:27:24'  
    ContentTime: '11:27:24'  
    Modality: 'CT'  
    ConversionType: 'WSD'  
    Manufacturer: 'GE MEDICAL SYSTEMS'  
    InstitutionName: 'JFK IMAGING CENTER'  
    ReferringPhysicianName: [1x1 struct]  
        StationName: 'CT01OC0'  
        StudyDescription: 'RT ANKLE'  
    PhysicianReadingStudy: [1x1 struct]  
        OperatorName: [1x1 struct]  
    ManufacturerModelName: 'GENESIS_ZEUS'  
    PatientGroupLength: 18  
        PatientName: [1x1 struct]  
    AcquisitionGroupLength: 10  
        SoftwareVersion: '03'  
    RelationshipGroupLength: 134  
        StudyInstanceUID:  
'1.2.840.113619.2.1.1.322987881.621.736170080.681'  
        SeriesInstanceUID:  
'1.2.840.113619.2.1.2411.1031152382.365.736169244'  
        SeriesNumber: 365  
        InstanceNumber: 1  
    ImagePresentationGroupLength: 168  
        SamplesPerPixel: 1  
    PhotometricInterpretation: 'MONOCHROME2'  
        Rows: 512  
        Columns: 512  
        BitsAllocated: 16  
        BitsStored: 16  
        HighBit: 15  
    PixelRepresentation: 1  
    SmallestImagePixelValue: 0  
        PixelPaddingValue: 0  
            WindowCenter: 1024  
            WindowWidth: 4095  
        RescaleIntercept: -1024  
        RescaleSlope: 1  
        RescaleType: 'US'  
    PixelDataGroupLength: 524296
```

Funciones relacionadas

`dicomdict`, `dicomread`, `dicomwrite`, `dicomuid`

dicomlookup

Encontrar un atributo en el diccionario de datos DICOM

Sintaxis

```
name = dicomlookup(group, element)
[group, element] = dicomlookup(name)
```

Descripción

name = dicomlookup(group, element) busca un atributo en el diccionario de datos DICOM usando las etiquetas de grupo y elemento especificadas en las variables group y element y devuelve una cadena que contiene el nombre del atributo. Las variables group y element pueden contener un valor decimal o una cadena hexadecimal.

[group, element] = dicomlookup(name) busca el atributo especificado por la variable name en el diccionario de datos DICOM actual y devuelve las etiquetas asociadas con el atributo en las variables group y element.

Ejemplos de uso

Ejemplo 1: Buscar algunos nombres de atributos DICOM utilizando sus etiquetas de grupo y elemento.

```
name1 = dicomlookup('7FE0', '0010')
name2 = dicomlookup(40, 4)

name1 =
PixelData

name2 =
PhotometricInterpretation
```

Ejemplo 2: Buscar las etiquetas grupo y elemento asociadas con el atributo “TransferSyntaxUID”.

```
[group, element] = dicomlookup('TransferSyntaxUID')

group =
2

element =
16
```

Ejemplo 2: Examinar los metadatos de un archivo en formato DICOM, buscando un nombre de atributo en el archivo.

```
metadata = dicominfo('CT-MONO2-16-ankle.dcm');
metadata.(dicomlookup('0028', '0004'))  
  
ans =  
MONOCHROME2
```

Funciones relacionadas

`dicomdict`, `dicominfo`

dicomread

Lectura de una imagen DICOM

Sintaxis

```
X = dicomread(filename)
X = dicomread(info)
[X,map] = dicomread(...)
[X,map,alpha] = dicomread(...)
[X,map,alpha,overlays] = dicomread(...)
[...] = dicomread(filename, 'frames', v)
```

Descripción

`X = dicomread(filename)` lee los datos de imagen desde el archivo DICOM especificado en la variable `filename`. Para imágenes simples en escala de grises, `X` es un array de M-por-N. Para imágenes simples en color verdadero, `X` es un array de M-por-N-por-3. Las imágenes múltiples (archivo de imagen que contiene más de una imagen) son siempre arrays en 4-D.

`X = dicomread(info)` lee los datos de imagen desde la variable `info`, una estructura de metadatos DICOM producida con la función `dicominfo`.

`[X,map] = dicomread(...)` devuelve la imagen en la variable `X` y el mapa de color en `map`. Si `X` es una imagen en escala de grises o en color verdadero, la variable `map` estará vacía.

`[X,map,alpha] = dicomread(...)` devuelve la imagen en la variable `X`, el mapa de color en `map` y una matriz de canal alfa para `X` en la variable `alpha`. Los valores en `alpha` serán 0 si el pixel es opaco e índices de filas apuntando a `map` en caso contrario. Para usar `alpha` se debe sustituir el valor RGB en `map` por el valor en `X`. `alpha` tiene la misma altura y anchura que `X` y es de 4-D para imágenes múltiples.

`[X,map,alpha,overlays] = dicomread(...)` devuelve la imagen en la variable `X`, el mapa de color en `map`, una matriz de canal alfa para `X` en la variable `alpha` y cualquier superposición en la variable `overlays`. Cada superposición (*overlay*) es una imagen de 1-bit en blanco y negro con la misma altura y anchura que `X`. Si hay varias superposiciones

en el archivo la variable `overlays` será una imagen múltiple en 4-D. Si no hay superposiciones, `overlays` estará vacía.

[...] = `dicomread(filename, 'frames', v)` lee solo las imágenes especificadas en el vector `v`, que debe ser un escalar entero, un vector de enteros o la cadena de caracteres '`'all'`' que es su valor por defecto.

`x` puede ser `uint8`, `int8`, `uint16`, o `int16`. `map` debe ser de tipo `double`. `alpha` tiene el mismo tipo y tamaño que `x`. `overlays` es un array de tipo `logical`.

La función `dicomread` soporta la lectura de archivos DICOM en formatos de compresión JPEG-2000 tanto en su tipo reversible (*lossless*) como irreversible (*lossy*).

Ejemplos de uso

Ejemplo 1: Utilizar la función `dicomread` para recoger el array de datos y la matriz de mapa de color de un archivo DICOM y crear después un montaje con las imágenes leídas.

```
[x, map] = dicomread('US-PAL-8-10x-echo.dcm');  
montage(x, map, 'Size', [4 3]);
```

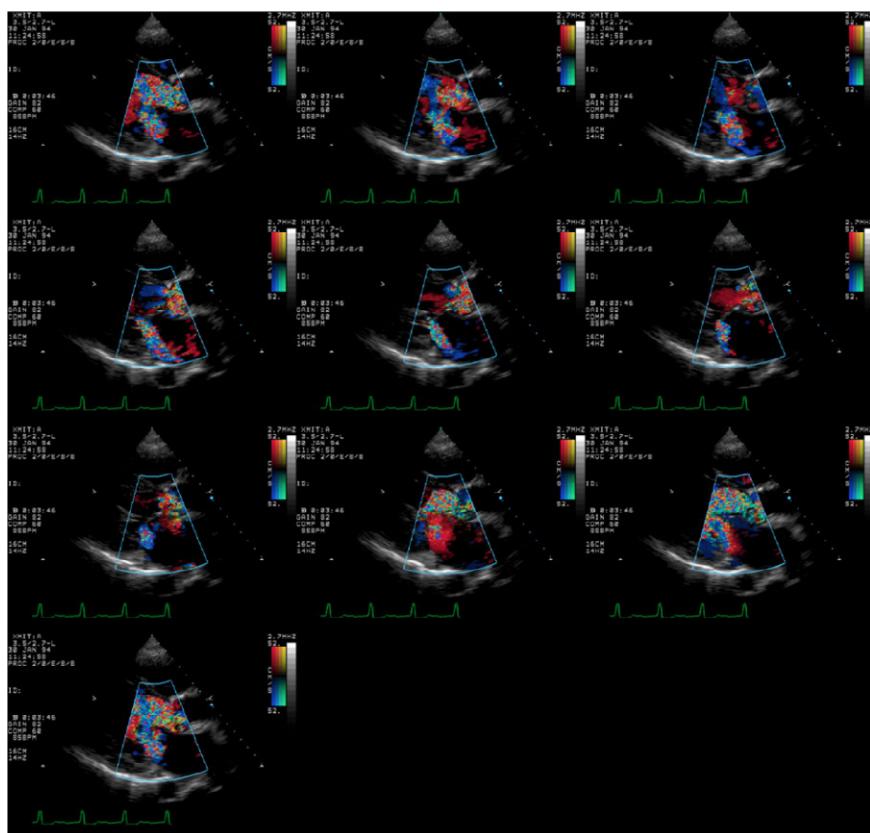


Figura f.126 - Montaje a partir de imagen DICOM

Ejemplo 2: Llamar a `dicomread` pasándole la información obtenida del archivo DICOM con `dicominfo` y mostrar después la imagen utilizando la función `imshow`. Ajustar el contraste con la herramienta `imcontrast`.

```
info = dicominfo('CT-MONO2-16-ankle.dcm');
I = dicomread(info);
imshow(I)
imcontrast;
```



Figura f.127 - Imagen original (I)



Figura f.128 - Imagen con contraste mejorado

Funciones relacionadas

`dicomdict`, `dicominfo`, `dicomwrite`

dicomuid

Generación de un identificador DICOM único

Sintaxis

```
UID = dicomuid
```

Descripción

`UID = dicomuid` crea una cadena `UID` que contiene un nuevo identificador DICOM único.

Si la función `dicomuid` recibe múltiples llamadas, esta produce valores únicos globales. Dos llamadas a la función siempre devuelven valores distintos.

Funciones relacionadas

`dicominfo`, `dicomwrite`

dicomwrite

Escritura de imágenes como archivos DICOM

Sintaxis

```
dicomwrite(X, filename)
dicomwrite(X, map, filename)
dicomwrite(..., param1, Valor1, param2, Valor2,...)
dicomwrite(..., 'ObjectType', IOD,...)
dicomwrite(..., 'SOPClassUID', UID,...)
dicomwrite(..., meta_struct,...)
dicomwrite(..., info,...)
status = dicomwrite(...)
```

Descripción

`dicomwrite(X, filename)` escribe la imagen binaria, en escala de grises o en color verdadero `X` al archivo nuevo DICOM cuyo nombre se especifica en la cadena `filename`.

`dicomwrite(X, map, filename)` escribe la imagen indexada `X` con su mapa de color `map`.

`dicomwrite(..., param1, Valor1, param2, Valor2,...)` especifica metadatos opcionales a escribir en el archivo DICOM o parámetros que afectarán a cómo se escribirá dicho archivo. `param1` es una cadena que contiene el nombre del atributo o una opción específica de la función `dicomwrite`. `Valor1` es el valor correspondiente al atributo u opción.

Para saber los atributos DICOM que se pueden especificar es preciso consultar el archivo de diccionario de datos DICOM de nombre `dicom-dict.txt`, incluido en la *toolbox*. La siguiente tabla explica las opciones que se pueden especificar, ordenadas alfabéticamente:

Opción	Descripción
'CompressionMode'	Cadena que especifica el tipo de compresión a usar cuando se guarda la imagen. Valores posibles: 'None' (por defecto) 'JPEG lossless' 'JPEG lossy' 'RLE'
'CreateMode'	Especifica el método utilizado para crear los datos que se guardarán en el nuevo archivo. Los valores posibles son:

	<p>'Create' - Comprueba los valores de entrada y genera los datos de entrada que faltan. (Por defecto)</p> <p>'Copy' - Copia todos los valores de entrada y no genera ningún dato que falta.</p>
'Dictionary'	Cadena que especifica el nombre de un diccionario de datos DICOM.
'Endian'	<p>Cadena que especifica el tipo de ordenación de byte del archivo: (más información sobre los tipos en la página de la función <code>analyze75info</code>)</p> <p>'Big'</p> <p>'Little' (por defecto)</p> <p>Nota: Si la variable VR vale 'Explicit', 'Endian' debe ser 'Big'. <code>dicomwrite</code> ignora este valor si se han especificado los parámetros 'CompressionMode' o 'TransferSintaxis'.</p>
'TransferSintaxis'	Especifica las opciones 'Endian', 'VR' y 'CompressionMode'.
	<p>Nota: Si se especifica, <code>dicomwrite</code> ignora cualquier valor de las opciones 'Endian', 'VR', y 'CompressionMode'.</p>
'VR'	<p>Cadena que especifica si el código VR debe escribirse al archivo.</p> <p>'explicit' - Escribir VR al archivo.</p> <p>'implicit' - Deducirlo desde el diccionario de datos. (Por defecto)</p> <p>Nota: Si se especifica el valor 'Endian' como 'Big', 'VR' debe ser 'Explicit'.</p>
'WritePrivate'	<p>Valor de tipo <code>Logical</code> que indica si los datos privados se deben escribir al archivo. Los valores posibles son:</p> <p><code>true</code> - Escribir datos privados al archivo.</p> <p><code>false</code> - No escribir datos privados. (Por defecto)</p>

`dicomwrite(..., 'ObjectType', IOD, ...)` escribe un archivo que contiene los metadatos necesarios para un tipo particular de Objeto de Información DICOM (IOD, *DICOM Information Object*). Los IODs permitidos son

- 'Secondary Capture Image Storage' (**por defecto**)
- 'CT Image Storage'
- 'MR Image Storage'

`dicomwrite(..., 'SOPClassUID', UID,...)` proporciona un método alternativo para especificar el IOD a crear. UID es el identificador único DICOM que corresponde con uno de los IODs listados anteriormente.

`dicomwrite(..., meta_struct,...)` especifica metadatos opcionales o opciones de archivo en la estructura `meta_struct`. Los nombres de los campos en la estructura `meta_struct` deben ser los nombres de los atributos/opciones del archivo DICOM. El valor de cada campo es el valor que se quiere asignar al atributo o a la opción.

`dicomwrite(..., info,...)` especifica metadatos en la estructura `info` generada con la función `dicominfo`. La estructura puede contener muchos campos. Para más información sobre esta estructura, consultar la página de la función `dicominfo`.

`status = dicomwrite(...)` devuelve información sobre los metadatos y la desencriptación utilizada para generar el archivo DICOM. Esta sintaxis puede ser útil cuando se especifica una estructura `info` creada por la función `dicominfo`. Si no se especifica ningún metadato, `dicomwrite` devuelve una matriz vacía (`[]`).

La estructura devuelta por `dicomwrite` contiene los siguientes campos:

Campo	Descripción
'BadAttribute'	La descripción interna del atributo está mal. No se debe poder encontrar desde el diccionario de datos o debe tener datos incorrectos en su descripción.
'MissingCondition'	El atributo es condicional pero no se ha especificado ninguna condición de cuándo usarlo.
'MissingData'	No se ha especificado ningún dato para un atributo necesario en el archivo.
'SuspectAttribute'	Los datos del atributo no coinciden con las especificaciones DICOM.

Nota: Las especificaciones del formato DICOM hablan de varias definiciones de objetos de información (IODs) que se pueden crear. Estas IODs corresponden a imágenes y metadatos producidos por modalidades distintas en la vida real (p.ej., MR, Rayos X, Ultrasonidos, etc.). Para cada tipo de IOD, las especificaciones DICOM definen el conjunto de metadatos que deben aparecer y los posibles valores de otros metadatos.

`dicomwrite` implementa un número limitado de estos IODs (listados arriba en el parámetro `ObjectType`) con los que verifica que todos los atributos están presentes, creando los atributos que faltan si es necesario. Sin embargo, si se quiere escribir archivos DICOM para IODs que no están implementados en `dicomwrite`, se debe utilizar el valor '`Copy`' del parámetro `CreateMode` para que `dicomwrite` escriba todos datos de imagen. De este modo se pueden coger los metadatos de un archivo del mismo tipo o IOD y usarlos para crear un nuevo archivo DICOM con diferentes datos de imagen. El inconveniente es que se corre el riesgo de copiar información no verificada por la función y que podría generar un archivo DICOM que no cumpliera con el estándar.

Ejemplos de uso

Ejemplo 1: Leer una imagen CT desde el archivo DICOM de ejemplo incluido con la *toolbox* y escribir la imagen en otro archivo, creando una segunda captura de la imagen.

```
X = dicomread('CT-MONO2-16-ankle.dcm');
dicomwrite(X, 'sc_file.dcm');
```

Ejemplo 2: Escribir la imagen X con sus metadatos en un nuevo archivo DICOM, usando la función `dicominfo` para obtener los metadatos de otro archivo DICOM.

```
metadata = dicominfo('CT-MONO2-16-ankle.dcm');
dicomwrite(X, 'ct_file.dcm', metadata);
```

Ejemplo 3: Copiar todos los metadatos desde un archivo a otro. De esta forma la función `dicomwrite` no verifica los metadatos escritos.

```
dicomwrite(X, 'ct_copy.dcm', metadata, 'CreateMode', 'copy');
```

Funciones relacionadas

`dicomdict`, `dicominfo`, `dicomread`, `dicomuid`

edge

Detección de bordes en una imagen en escala de grises

Sintaxis

```
BW = edge(I)

BW = edge(I,'sobel')
BW = edge(I,'sobel',thresh)
BW = edge(I,'sobel',thresh,direction)
[BW,thresh] = edge(I,'sobel',...)

BW = edge(I,'prewitt')
BW = edge(I,'prewitt',thresh)
BW = edge(I,'prewitt',thresh,direction)
[BW,thresh] = edge(I,'prewitt',...)

BW = edge(I,'roberts')
BW = edge(I,'roberts',thresh)
[BW,thresh] = edge(I,'roberts',...)

BW = edge(I,'log')
BW = edge(I,'log',thresh)
BW = edge(I,'log',thresh,sigma)
[BW,threshold] = edge(I,'log',...)

BW = edge(I,'zerocross',thresh,h)
[BW,thresh] = edge(I,'zerocross',...)

BW = edge(I,'canny')
BW = edge(I,'canny',thresh)
BW = edge(I,'canny',thresh,sigma)
[BW,threshold] = edge(I,'canny',...)
```

Introducción

La detección de bordes es la forma más utilizada en segmentación para encontrar discontinuidades en valores de intensidad. Se lleva a cabo mediante derivadas de primer y segundo orden y la IPT cuenta con una función específica para ello, la función `edge`, desde la que se puede detectar los bordes de una imagen utilizando diferentes métodos.

Descripción

`BW = edge(I)` recibe como entrada una imagen binaria o en escala de grises `I` y devuelve una imagen binaria `BW` del mismo tamaño que `I`, con 1's donde la función haya encontrado bordes y 0's en el resto. `I` es un array disperso tipo `numeric`. `BW` es de tipo `logical`.

`edge` utiliza por defecto el método Sobel para detectar los bordes, pero se pueden utilizar otros métodos que se listan a continuación:

- El método Sobel detecta los bordes usando la aproximación Sobel. Suele obtener buenos resultados con las opciones por defecto. (**Por defecto**)
- El método Prewitt detecta los bordes usando la aproximación Prewitt. Suele obtener resultados parecidos a los obtenidos con el método Sobel.
- El método Roberts detecta los bordes usando la aproximación Roberts. Uno de los métodos más simples y antiguos de detección de bordes. No suele usarse mucho salvo que se necesite simpleza y rapidez.
- El método del Laplaciano del Gaussiano detecta los bordes buscando cortes con cero después de filtrar I con un filtro Laplaciano o Gaussiano. Obtiene normalmente mejores resultados que el método Sobel pero a costa de tener que modificar los parámetros por defecto.
- El método de cruce por cero detecta los bordes buscando cruces con cero después de filtrar I con el filtro que se especifique. Está basado en el mismo concepto que el método del Laplaciano del Gaussiano pero se utiliza un filtro especificado para realizar la convolución.
- El método Canny detecta los bordes buscando máximos locales del gradiente de I . Este método es el más potente de todos y por ello es el más utilizado. Es menos propenso a confundirse por el ruido y más propenso a detectar bordes débiles. Para obtener el mejor resultado normalmente se necesitan modificar sus parámetros por defecto.

Normalmente los métodos de Sobel, Prewitt y Roberts detectan menos bordes de los necesarios utilizando las opciones por defecto (utilizan mayor umbral de detección) mientras que los métodos del Laplaciano del Gaussiano, de cruce por cero y de Sobel suelen detectar más bordes de los necesarios por defecto (menor umbral).

Los parámetros que se pueden utilizar dependen del tipo de método elegido aunque son bastante parecidos en todos ellos:

Método Sobel (por defecto)

`BW = edge(I, 'sobel')` especifica que se usará el método Sobel.

`BW = edge(I,'sobel',thresh)` especifica el umbral de sensibilidad del método Sobel en la variable `thresh`. `edge` ignora todos los bordes que no son más fuertes que el umbral especificado. Si no se especifica la variable `thresh`, o `thresh` está vacía ([]), `edge` elige su valor automáticamente.

`BW = edge(I,'sobel',thresh,direction)` especifica la dirección de detección del método Sobel en la variable `direction`, una cadena que indica si se debe buscar por bordes horizontales ('horizontal'), verticales ('vertical') o ambos ('both') (por defecto).

`BW = edge(I,'sobel',...,options)` especifica opciones extra. La cadena 'nothinning' acelera la operación del algoritmo saltando la etapa adicional de *thinning* o adelgazamiento del borde. Cuando se especifica la cadena 'thinning' (por defecto) el algoritmo aplica *thinning* o adelgazamiento del borde.

`[BW,thresh] = edge(I,'sobel',...)` devuelve el valor del umbral.

`[BW,thresh,gv,gh] = edge(I,'sobel',...)` devuelve bordes verticales y horizontales de respuesta a los operadores de Sobel.

Método Prewitt

`BW = edge(I,'prewitt')` especifica que se usará el método de Prewitt.

`BW = edge(I,'prewitt',thresh)` especifica el umbral de sensibilidad del método Prewitt en la variable `thresh`. `edge` ignora todos los bordes que no son más fuertes que `thresh`. Si no se especifica la variable `thresh`, o `thresh` está vacía ([]), `edge` elige su valor automáticamente.

`BW = edge(I,'prewitt',thresh,direction)` especifica la dirección de detección del método Prewitt en la variable `direction`, una cadena que indica si se debe buscar por bordes horizontales ('horizontal'), verticales ('vertical') o ambos ('both') (por defecto).

`[BW,thresh] = edge(I,'prewitt',...)` devuelve el valor del umbral.

Método Roberts

`BW = edge(I, 'roberts')` especifica que se usará el método de Roberts.

`BW = edge(I, 'roberts', thresh)` especifica el umbral de sensibilidad del método Roberts en la variable `thresh`. `edge` ignora todos los bordes que no son más fuertes que `thresh`. Si no se especifica la variable `thresh`, o `thresh` está vacía ([]), `edge` elige su valor automáticamente.

`BW = edge(I, 'roberts', ..., options)` especifica opciones extra. La cadena '`nothinning`' acelera la operación del algoritmo saltando la etapa adicional de *thinning* o adelgazamiento del borde. Cuando se especifica la cadena '`thinning`' (por defecto) el algoritmo aplica *thinning* o adelgazamiento del borde.

`[BW, thresh] = edge(I, 'roberts', ...)` devuelve el valor del umbral.

`[BW, thresh, g45, g135] = edge(I, 'roberts', ...)` devuelve respuestas de bordes de 45 grados y 135 grados a los operadores Roberts.

Método Laplaciano del Gaussiano

`BW = edge(I, 'log')` especifica que se usará el método Laplaciano del Gaussiano.

`BW = edge(I, 'log', thresh)` especifica el umbral de sensibilidad del método Laplaciano del Gaussiano en la variable `thresh`. `edge` ignora todos los bordes que no son más fuertes que `thresh`. Si no se especifica la variable `thresh`, o `thresh` está vacía ([]), `edge` elige su valor automáticamente. Si se especifica un umbral de 0, la imagen de salida tendrá contornos cerrados porque incluirá todos los cruces por cero de la imagen de entrada.

`BW = edge(I, 'log', thresh, sigma)` especifica que se usará el método Laplaciano del Gaussiano usando la variable `sigma` como la desviación estándar del filtro LoG (Laplaciano del Gaussiano). El valor por defecto de `sigma` es 2; el tamaño del filtro es `n`-por-`n`, donde `n = ceil(sigma*3)*2+1`.

`[BW, thresh] = edge(I, 'log', ...)` devuelve el valor del umbral.

Método de Cruce por cero

`BW = edge(I, 'zerocross', thresh, h)` especifica que se usará el método de Cruce por cero usando el filtro `h` con el umbral definido en la variable `thresh`; si el argumento está vacío ([]), `edge` elige el umbral automáticamente. Si el umbral es 0, la imagen de salida tendrá contornos cerrados porque incluirá todos los cruces por cero de la imagen de entrada.

`[BW, thresh] = edge(I, 'zerocross', ...)` devuelve el valor del umbral.

Método Canny

`BW = edge(I, 'canny')` especifica que se usará el método Canny.

`BW = edge(I, 'canny', thresh)` especifica los umbrales de sensibilidad para el método Canny en la variable `thresh`, un vector de dos elementos en el que el primer elemento es el umbral bajo y el segundo el alto. Si se especifica un escalar en la variable `thresh`, este valor se usará para el umbral alto y `0.4*thresh` para el umbral bajo. Si no se usa la variable `thresh`, o `thresh` está vacía ([]), `edge` elige su valor automáticamente. El valor para `thresh` es relativo al valor más alto de la magnitud del gradiente de la imagen.

`BW = edge(I, 'canny', thresh, sigma)` especifica que se utilizará el método Canny usando `sigma` como la desviación estándar del filtro Gaussiano. El valor por defecto de `sigma` es 1; el tamaño del filtro se elige automáticamente basándose en la variable `sigma`.

`[BW, thresh] = edge(I, 'canny', ...)` devuelve el valor del umbral.

Nota: En todos los casos el umbral por defecto se elige de forma heurística según los datos de entrada. La mejor manera de corregir el umbral es ejecutar la función `edge` capturando el umbral de salida y posteriormente ajustar con un umbral mayor (menos píxeles de borde) o menor (más píxeles de borde).

Ejemplos de uso

Ejemplo 1: Encontrar los bordes horizontales y verticales de la imagen comparando los resultados obtenidos por los métodos de Sobel y Prewitt con las opciones por defecto.

```
I = imread('casa.tif');
imshow(I)

BWHS = edge(I,'sobel','horizontal'); figure, imshow(BWHS)
BWHP = edge(I,'prewitt','horizontal'); figure, imshow(BWHP)
```

```
BWVS = edge(I,'sobel','vertical'); figure, imshow(BWVS)
BWVP = edge(I,'prewitt','vertical'); figure, imshow(BWVP)
```



Figura f.129 - Imagen original (I)



Figura f.130 - Bordes horizontales con el método Sobel (BWHS)



Figura f.131 - Bordes horizontales con el método Prewitt (BWHP)



Figura f.132 - Bordes verticales con el método Sobel (BWVS)



Figura f.133 - Bordes verticales con el método Prewitt (BWVP)

Vemos como los resultados son prácticamente idénticos con ambos métodos.

Ejemplo 2: Encontrar los bordes de una imagen usando todos los métodos por defecto de la función `edge` y comparar los resultados variando las opciones por defecto para obtener mejor resultado de los bordes principales.

```
I = imread('casa.tif');
imshow(I)
```

```
[BW1d,umbral1def] = edge(I,'sobel'); figure, imshow(BW1d)
BW1 = edge(I,'sobel',0.05); figure, imshow(BW1)

[BW2d,umbral2def] = edge(I,'prewitt'); figure, imshow(BW2d)
BW2 = edge(I,'prewitt',0.05); figure, imshow(BW2)

[BW3d,umbral3def] = edge(I,'roberts'); figure, imshow(BW3d)
BW3 = edge(I,'roberts',0.06); figure, imshow(BW3);

[BW4d,umbral4def] = edge(I,'log'); figure, imshow(BW4d)
BW4 = edge(I,'log',0.004); figure, imshow(BW4)

[BW5d,umbral5def] = edge(I,'zerocross'); figure, imshow(BW5d)
BW5 = edge(I,'zerocross',0.0045); figure, imshow(BW5)

[BW6d,umbral6def] = edge(I,'canny'); figure, imshow(BW6d)
BW6 = edge(I,'canny',[0.02 0.065],1.7); figure, imshow(BW6)
```



Figura f.129 - Imagen original (*I*)



Figura f.134 - Bordes (BW1d) con el método Sobel por defecto



Figura f.135 - Bordes (BW1) con el método Sobel disminuyendo el umbral por defecto



Figura f.136 - Bordes (BW2d) con el método Prewitt por defecto



Figura f.137 - Bordes (BW2) con el método Prewitt disminuyendo el umbral por defecto



Figura f.138 - Bordes (BW3d) con el método Roberts por defecto



Figura f.139 - Bordes (BW3) con el método Roberts disminuyendo el umbral por defecto



Figura f.140 - Bordes (BW4d) con el método LoG por defecto



Figura f.141 - Bordes (BW4) con el método LoG por aumentando el umbral por defecto



Figura f.142 - Bordes (BW5d) con el método Zerocross por defecto



Figura f.143 - Bordes (BW5) con el método Zerocross aumentando el umbral por defecto



Figura f.144 - Bordes (BW6d) con el método Canny por defecto



Figura f.145 - Bordes (BW6) con el método Canny aumentando el umbral por defecto

Ejemplo 3: Encontrar los bordes de una imagen médica de una bacteria.

```
I = imread('ecoli.tif');
imshow(I)
BW = edge(I, 'canny', [0.15 0.25], 1.9);
figure, imshow(BW)
```

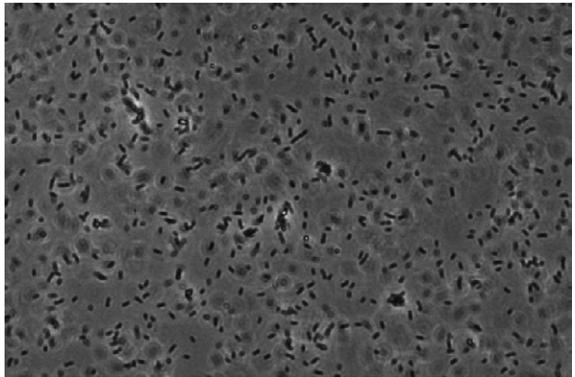


Figura f.146 - Imagen original (I)

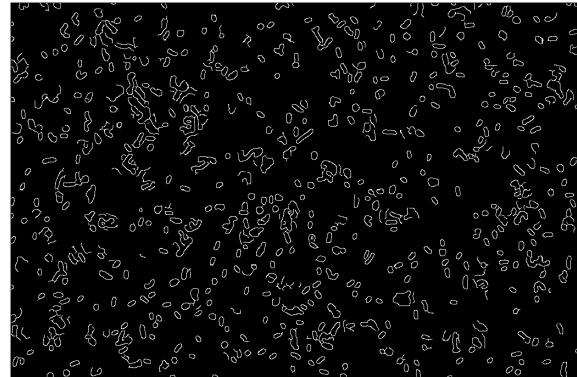


Figura f.147 - Bordes (BW) con el método Canny

Funciones relacionadas

`fspecial`

edgetaper

Reducción de las discontinuidades alrededor de los bordes de una imagen

Sintaxis

```
J = edgetaper(I,PSF)
```

Descripción

`J = edgetaper(I,PSF)` desenfoca los bordes de la imagen de entrada `I` usando la función de dispersión del punto `PSF`. El tamaño de `PSF` no puede exceder la mitad del tamaño de la imagen en ninguna dimensión.

La imagen de salida `J` es la suma ponderada de la imagen original y su versión desenfocada (la función `edgetaper` elimina las altas frecuencias en el borde de la imagen desenfocando la imagen completa y posteriormente reemplazando los píxeles centrales por los originales. De este modo, los bordes de la imagen se estrechan hasta baja frecuencia). El array de ponderación, determinado por la función de autocorrelación de `PSF`, hace a `J` igual que `I` en su región central, e igual que su versión desenfocada cerca de los bordes.

La función `edgetaper` reduce el *ringing* (artefactos oscilantes en imágenes restauradas) en los métodos de desenfoque que usan la transformada discreta de Fourier como las funciones `deconvwnr`, `deconvreg`, y `deconvlucy`. Para ello se aplica la función antes de aplicarle el desenfoque.

`I` y `PSF` pueden ser del tipo `uint8`, `uint16`, `int16`, `single`, o `double`. `J` es del mismo tipo que `I`.

Ejemplos de uso

Ejemplo 1: Desenfocar los bordes de una imagen.

```
I = imread('moon.tif');
PSF = fspecial('gaussian',60,10);
ET = edgetaper(I,PSF);
figure, imshow(I,[ ])
figure, imshow(ET,[ ])
```

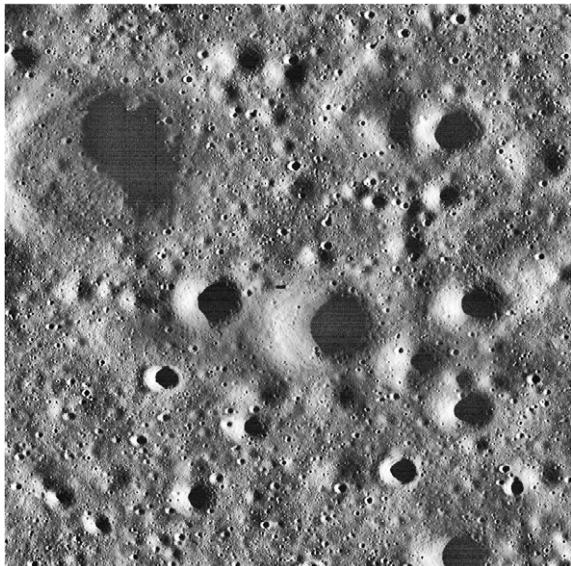


Figura f.148 - Imagen original (I)

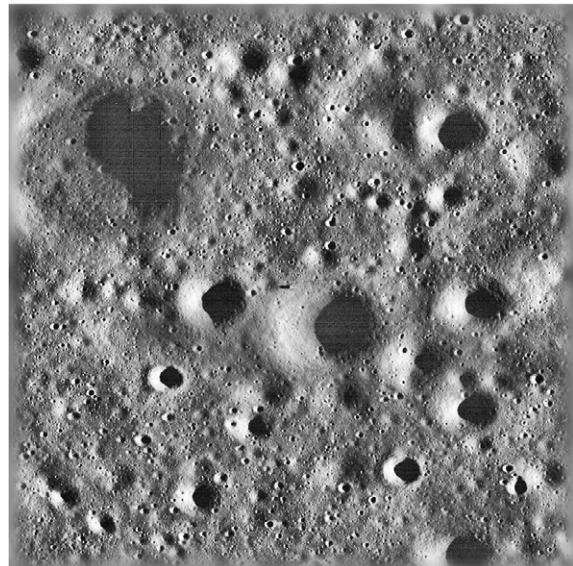


Figura f.149 - Imagen procesada (ET)

Funciones relacionadas

`deconvlucy`, `deconvreg`, `deconvwnr`, `otf2psf`, `padarray`, `psf2otf`

entropy

Entropía de una imagen en escala de grises

Introducción

La IPT dispone de dos funciones específicas para el cálculo de la entropía de una imagen: `entropy` y `entropyfilt`. `entropy` calcula la entropía de una imagen completa mientras que `entropyfilt` calcula la entropía de las regiones de una imagen.

La entropía de una imagen es una medida de su información. Si la entropía es alta, la información tiende a ser altamente imprevisible porque algo imprevisible o infrecuente aporta más información que algo previsible. Una imagen con información de alta entropía, contiene mucha aleatoriedad y baja redundancia.

Dentro del campo de la compresión de imágenes, la entropía mide la media de información necesaria para codificar los valores de la imagen. En relación con el histograma de una imagen, la entropía es una medida de la dispersión del histograma. Y en segmentación existen métodos de umbralización de imágenes en escala de grises basados en la maximización de la entropía y es frecuente utilizar la entropía para caracterizar la textura de una imagen.

Sintaxis

```
E = entropy(I)
```

Descripción

`E = entropy(I)` devuelve un escalar `E` que representa la entropía de la imagen en escala de grises `I`.

`I` puede ser una imagen multidimensional. Si `I` tiene más de dos dimensiones, como las imágenes RGB, la función `entropy` la trata como una imagen multidimensional en escala de grises y no como una imagen RGB. `I` puede ser de tipo `logical`, `uint8`, `uint16`, o `double` y debe ser real, no nula y no dispersa. `E` es de tipo `double`.

Ejemplos de uso

Ejemplo 1: Comparar la entropía y el histograma de una imagen con el histograma normal y la misma imagen con el histograma recortado.

```
I1 = imread('rueda.tif'); imshow(I1)
figure, imhist(I1)
E1 = entropy(I1)

I2 = imread('rueda2.tif'); figure, imshow(I2)
figure, imhist(I2)
E2 = entropy(I2)

E1 =
    7.7630

E2 =
    6.7257
```



Figura f.28 - Imagen original (I1)



Figura f.150 - Imagen histograma recortado (I2)

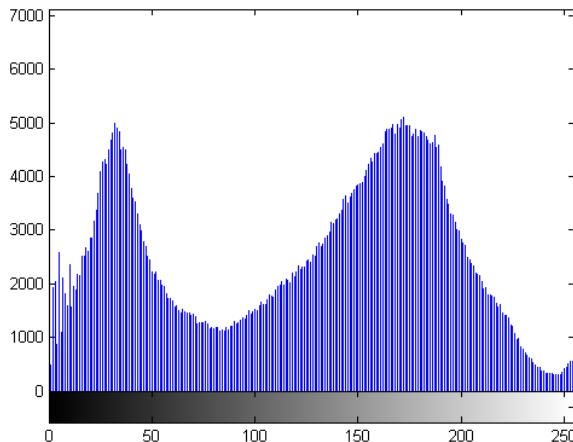


Figura f.151 - Histograma de imagen original II

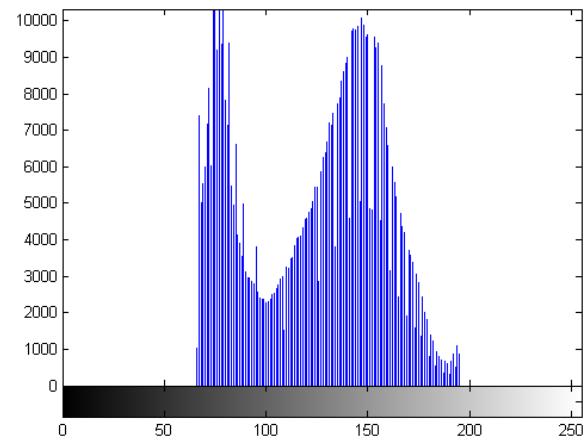


Figura f.152 - Histograma de imagen I2

Funciones relacionadas

imhist, entropyfilt

entropyfilt

Entropía regional de una imagen en escala de grises

Introducción

La entropía de una imagen es una medida de su información. La IPT dispone de dos funciones específicas para el cálculo de la entropía de una imagen: `entropy` y `entropyfilt`. `entropy` calcula la entropía de una imagen completa mientras que `entropyfilt` calcula la entropía de las regiones de una imagen.

Uno de los usos comunes de la medida de la entropía de una imagen por regiones es que puede utilizarse para caracterizar la textura de la imagen. Existen tres funciones de análisis de textura (`rangefilt`, `stdfilt` y `entropyfilt`) que utilizan medidas estadísticas estándar como son el rango, la desviación estándar y la entropía. Estas medidas pueden caracterizar la textura de una imagen porque proporcionan información a cerca de la variabilidad local de la intensidad de los píxeles. Estas funciones se suelen utilizar cuando los objetos de la imagen están más caracterizados por textura que por intensidad ^[13], por ejemplo en técnicas de segmentación de texturas en las que se busca identificar regiones basándose en su textura.

Sintaxis

```
J = entropyfilt(I)
J = entropyfilt(I,NHOOD)
```

Descripción

`J = entropyfilt(I)` devuelve el array `J`, donde cada pixel de salida contiene la entropía de la región de 9-por-9 alrededor del pixel correspondiente en la imagen de entrada `I`.

`J = entropyfilt(I,NHOOD)` realiza filtrado de entropía a la imagen de entrada `I` especificando la región a procesar en la variable `NHOOD`. `NHOOD` es un array multidimensional de unos y ceros a modo de patrón donde los valores distintos de cero especifican la región. El tamaño de `NHOOD` debe ser impar en cada dimensión.

Por defecto, `entropyfilt` usa la región `true(9)`. `entropyfilt` determina el elemento central de la región con `floor((size(NHOOD) + 1)/2)`. Para especificar regiones de tamaños diferentes como por ejemplo un disco, usar la función `strel` para crear un patrón o elemento estructurante y después usar la función `getnhood` para extraer la región desde dicho elemento en un array.

`I` puede ser de cualquier dimensión. Si `I` tiene más de dos dimensiones, como las imágenes RGB, la función `entropy` la trata como una imagen multidimensional en escala de grises y no como una imagen RGB. La imagen de salida `J` es del mismo tamaño que la de entrada `I` y suele denominarse imagen textura.

`I` puede ser de tipo `logical`, `uint8`, `uint16`, o `double` y debe ser real y no disperso. `NHOOD` puede ser de tipo `numeric` o `logical` y debe contener ceros o unos. El array de salida `J` es de tipo `double`.

Ejemplos de uso

Ejemplo 1: Aplicar el filtro de entropía por defecto.

```
I = imread('circuit.tif');
J = entropyfilt(I);
imshow(I), figure, imshow(J,[])
```

Ejemplo 2: Comparar los resultados obtenidos con las tres funciones de análisis de textura:

`rangefilt`, `stdfilt` y `entropyfilt`

% Leemos la imagen original y la mostramos.

```
A=imread('euros.tif');
subplot(2,2,1), imshow(I), title('original')
```

% Aplicamos filtro de desviación estándar y mostramos resultado en el rango [min,max].

```
J = stdfilt(I);
subplot(2,2,2),imshow(J,[]),title('stdfilt')
```

% Aplicamos filtro de entropía en una región de 15x15 y mostramos resultado en el rango [min,max].

```
J = entropyfilt(I,ones(15));
subplot(2,2,3),imshow(J,[]),title('entropyfilt')
```

% Aplicamos filtro de rango en una región de 5x5 y mostramos resultado en el rango [min,max].

```
J = rangefilt(I,ones(5));
subplot(2,2,4),imshow(J,[ ]),title('entropyfilt')
```

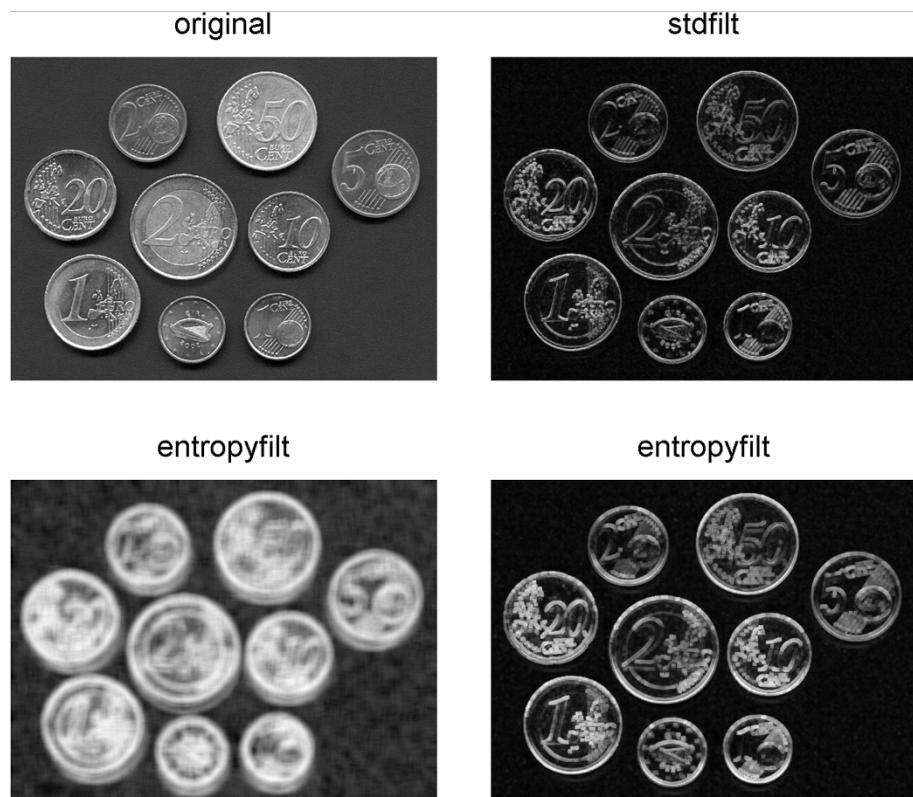


Figura f.153 - Comparación funciones análisis de textura

Ejemplo 3: Crear una máscara de las texturas de una imagen para luego segmentarlas.

```
I = imread('texture.tif');
imshow(I)

% Creamos la imagen textura a partir del filtro de entropía y la
reescalamos para convertirla en una imagen tipo double.

E = entropyfilt(I);
Eim = mat2gray(E);
figure, imshow(Eim)

% Destacamos ambas texturas convirtiendo la imagen a formato binario
utilizando un nivel específico que permita diferenciarlas bien.

BW1 = im2bw(Eim, .73);
figure, imshow(BW1)

% Extraemos la textura más definida con la función bwareaopen.

BWao = bwareaopen(BW1,2000);
figure, imshow(BWao)

% Suavizamos bordes y cerramos huecos.

nhood = true(9);
closeBWao = imclose(BWao,nhood);
roughMask = imfill(closeBWao,'holes');
```

% Mostramos la máscara rápida final.

```
figure, imshow(roughMask)
```

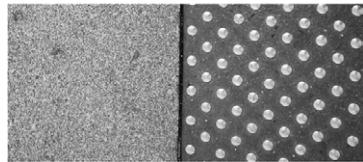


Figura f.154 - Imagen original I

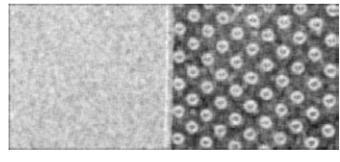


Figura f.155 - Imagen textura

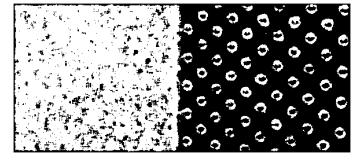


Figura f.156 - Imagen binaria

BW1



Figura f.157 - Extracción de
textura



Figura f.158 - Máscara final

Funciones relacionadas

entropy, imhist, rangefilt, stdfilt

fan2para

Conversión de proyecciones en abanico en proyecciones paralelas

Introducción

Una proyección está formada por la combinación de un conjunto de integrales de línea. Existen dos tipos de proyecciones principales: proyecciones paralelas (o proyecciones de eje paralelo) y proyecciones en abanico.

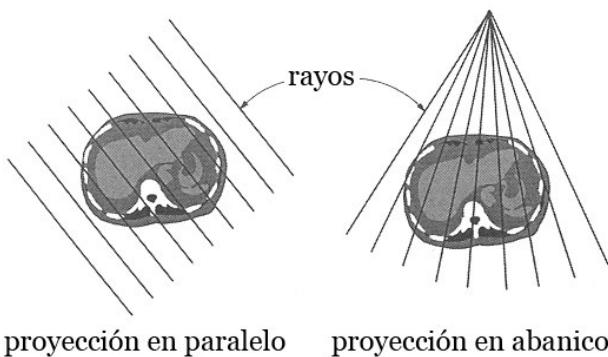


Figura f.159 - Proyecciones

La base matemática para la reconstrucción a partir de una serie de proyecciones es la Transformada de Radon, una transformación integral que consiste en la integral sobre un conjunto de rectas. La Transformada de Radon $R\{f(x,y)\}$ de una imagen representada por la función $f(x,y)$ es una serie de integrales de línea a través $f(x,y)$ con diferentes ángulos de rotación θ respecto al origen (proyección en paralelo de la intensidad de la imagen a lo largo de una línea radial orientada con un ángulo específico):

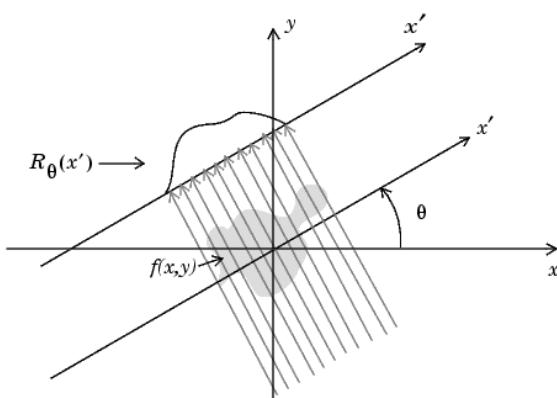


Figura f.160 - Transformada de Radon para un ángulo θ fijo

Sintaxis

```
P = fan2para(F,D)
P = fan2para(..., param1, val1, param2, val2,...)
[P ,parallel_locations, parallel_rotation_angles] = fan2para( ...)
```

Descripción

`P = fan2para(F,D)` convierte los datos de haz en abanico `F` a datos de haz en paralelo `P`. Cada columna de `F` contiene las muestras tomadas por el sensor del haz en abanico a un único ángulo de rotación. `D` es la distancia en píxeles desde el vértice del haz en abanico al centro de rotación que se usó para obtener las proyecciones.

`P = fan2para(..., param1, val1, param2, val2,...)` especifica parámetros que controlan varios aspectos de la conversión, detallados a continuación. Los nombres de los parámetros se pueden abbreviar y no son sensibles a las mayúsculas:

Parámetro	Descripción
'FanCoverage'	Cadena que especifica la cobertura de rotación de los ejes: 'cycle' — Rotar completamente [0, 360). (Por defecto) 'minimal' — Rotar lo mínimo necesario para representar el objeto.
'FanRotationIncrement'	Escalar real positivo que especifica el incremento del ángulo de rotación de las proyecciones de haz en abanico. Se mide en grados. (Por defecto : 1)
'FanSensorGeometry'	Cadena que especifica cómo se posicionan los sensores: 'arc' — Los sensores están equiespaciados a través de un arco circular a una distancia <code>D</code> del centro de rotación. (Por defecto)

Figura f.16I - Sensores en arco

	<p>'line' — Los sensores están equiespaciados a lo largo de una línea.</p>
	<p><i>Figura f.162 - Sensores en línea</i></p>
'FanSensorSpacing'	<p>Escalar real positivo que especifica el espaciado de los sensores de haz en abanico. La interpretación del valor depende de la variable 'FanSensorGeometry' .</p> <p>Si 'FanSensorGeometry' se pone a 'arc' (por defecto) el valor define el espaciado angular en grados (por defecto: 1)</p> <p>Si 'FanSensorGeometry' vale 'line', el valor define el espaciado lineal (por defecto: 1). Consultar la función fanbeam para más detalle.</p> <p>Nota: Este espaciado lineal se mide en el eje x. El origen de los ejes es el pixel central de la imagen.</p>
'Interpolation'	<p>Cadena de texto que especifica el tipo de interpolación que se usará entre los datos de haz paralelo y haz en abanico.</p> <ul style="list-style-type: none"> 'nearest' — Vecino más cercano {'linear'} — Lineal 'spline' — Spline cúbico a trozos 'pchip' — Hermite (PCHIP) cúbico a trozos 'cubic' — Igual que 'pchip'
'ParallelCoverage'	<p>Cadena de texto que especifica la cobertura de rotación.</p> <ul style="list-style-type: none"> 'cycle' — El dato paralelo cubre 360 grados {'halfcycle'} — El dato paralelo cubre 180 grados
'ParallelRotationIncrement'	<p>Escalar real positivo que especifica el incremento del ángulo de rotación de las proyecciones de haz en paralelo. Se mide en grados. Los ángulos del haz en paralelo se calculan para cubrir [0,180) grados con incrementos de PAR_ROT_INC que es el valor de 'ParallelRotationIncrement'. 180/PAR_ROT_INC debe ser un entero.</p> <p>Si no se especifica la variable, se asume el mismo incremento que el incremento de los ángulos de rotación del</p>

	haz en abanico.
'ParallelSensorSpacing'	<p>Escalar real positivo que especifica en píxeles el espaciado de los sensores de haz en paralelo. El rango de las localizaciones del sensor está sujeto al rango de los ángulos en abanico y viene dado por:</p> $[D \cdot \tan(\min(FAN_ANGLES)), \dots, D \cdot \tan(\max(FAN_ANGLES))]$ <p>Si no se especifica la variable, se asume un espaciado uniforme y se pone al mínimo espaciado sujeto a los ángulos en abanico y muestreando por encima del rango sujeto a los ángulos en abanico.</p>

[P, parallel_locations, parallel_rotation_angles] = fan2para(...) devuelve las localizaciones del sensor de haz paralelo en la variable parallel_locations y los ángulos de rotación donde se calcularon las proyecciones en la variable parallel_rotation_angles.

Los parámetros de entrada, F y D, pueden ser de tipo double o single y deben ser no dispersos. Cualquier otra entrada numérica es de tipo double. La salida P es double.

Ejemplos de uso

Ejemplo 1: Crear datos de haz en paralelo a partir de una imagen de una “cabeza de práctica” o *phantom head* que se suele utilizar para probar algoritmos de reconstrucción en dos dimensiones como la Transformada de Radón. Convertir a datos de haz en abanico y recuperar los datos de haz en paralelo originales.

```
% Creamos una imagen de "cabeza de práctica" (phantom head).
ph = phantom(128);
theta = 0:179;

% Hacemos la Transformada de Radon de la imagen y recogemos las
coordenadas radiales correspondientes a cada fila (valores a lo largo del
eje x' orientado theta grados según el eje x). La mostramos.

[Psynthetic,xp] = radon(ph,theta);
imshow(Psynthetic,[],'XData',theta,'YData',xp,'InitialMagnification','fit')
axis normal
title('Datos de haz en paralelo')
xlabel('\theta (grados)')
ylabel('x''')
colormap(hot), colorbar

% Convertimos los datos de haz en paralelo en haz en abanico.
```

```
Fsynthetic = para2fan(Psynthetic,100,'FanSensorSpacing',1);

% Recuperamos los datos de haz en paralelo originales convirtiendo datos
de haz en abanico en paralelo. Mostramos para comparar con original.

[Precovered,Ploc,Pangles] =
fan2para(Fsynthetic,100,'FanSensorSpacing',1,'ParallelSensorSpacing',1);
figure
imshow(Precovered,[],'XData',Pangles,'YData',Ploc,'InitialMagnification',
'fit')
axis normal
title('Datos de haz en paralelo recuperados')
xlabel('Ángulos de rotación (grados)')
ylabel('Localizaciones del sensor paralelo (pixeles)')
colormap(hot), colorbar
```

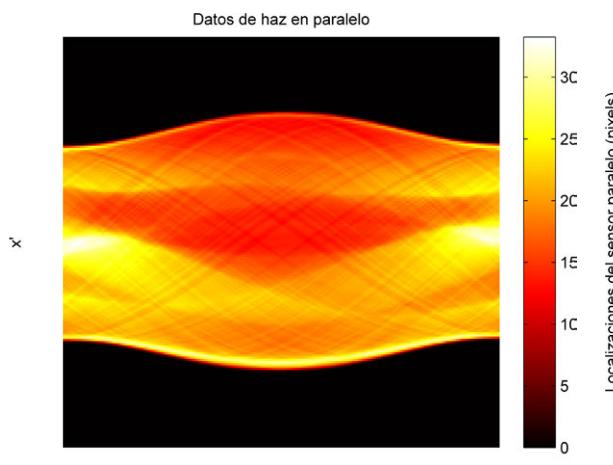


Figura f.163 - Datos de haz en paralelo originales

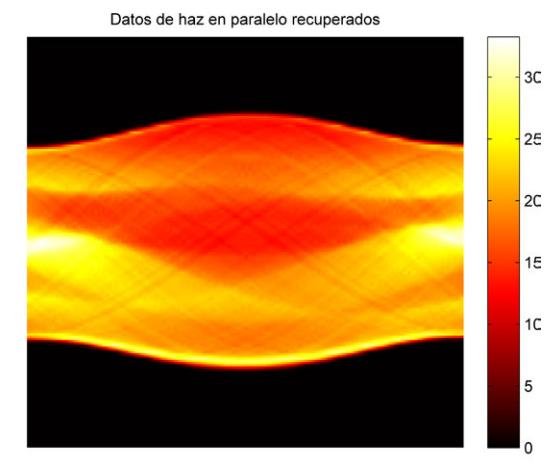


Figura f.164 - Datos de haz en paralelo recuperados

Funciones relacionadas

fanbeam, ifanbeam, iradon, para2fan, phantom, radon

fanbeam

Transformación (proyecciones) de haz en abanico

Introducción

Una proyección está formada por la combinación de un conjunto de integrales de línea. Existen dos tipos de proyecciones principales: proyecciones paralelas (o proyecciones de eje paralelo) y proyecciones en abanico.

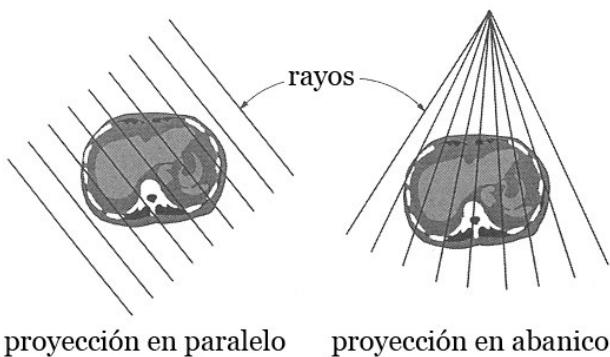


Figura f.159 - Proyecciones

La base matemática para la reconstrucción a partir de una serie de proyecciones es la Transformada de Radon, una transformación integral que consiste en la integral sobre un conjunto de rectas. La Transformada de Radon $R\{f(x,y)\}$ de una imagen representada por la función $f(x,y)$ es una serie de integrales de línea a través $f(x,y)$ con diferentes ángulos de rotación θ respecto al origen (proyección en paralelo de la intensidad de la imagen a lo largo de una línea radial orientada con un ángulo específico):

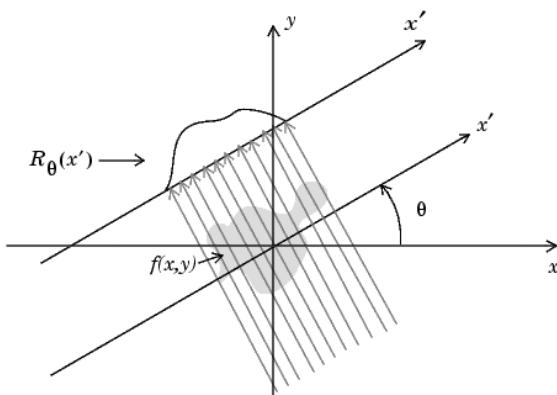


Figura f.160 - Transformada de Radon para un ángulo θ fijo

Puede entonces construirse una colección bidimensional ordenada de proyecciones en la cual uno de los ejes corresponde a la posición y el otro al ángulo. Esta estructura recibe el

nombre de sinograma. Su nombre se debe a la variación sinusoidal de la posición del objeto dentro de las proyecciones, debido al cambio consecutivo en el ángulo de rotación. El nombre suele utilizarse en tomografía para referirse a la transformada de Radon.^[14]

Sintaxis

```
F = fanbeam(I,D)
F = fanbeam(..., param1, val1, param1, val2,...)
[F, fan_sensor_positions, fan_rotation_angles] = fanbeam(...)
```

Descripción

`F = fanbeam(I,D)` calcula los datos de haz en abanico (sinograma) `F` a partir de la imagen `I`. `D` es la distancia en píxeles desde el vértice de haz en abanico al centro de la rotación, que es el pixel central de la imagen y está definido como `floor((size(I)+1)/2)`. `D` debe ser suficientemente largo para asegurar que el vértice está fuera de la imagen en todos los ángulos de rotación. La siguiente figura muestra `D` en relación al vértice del haz en abanico para una geometría de haz en abanico:

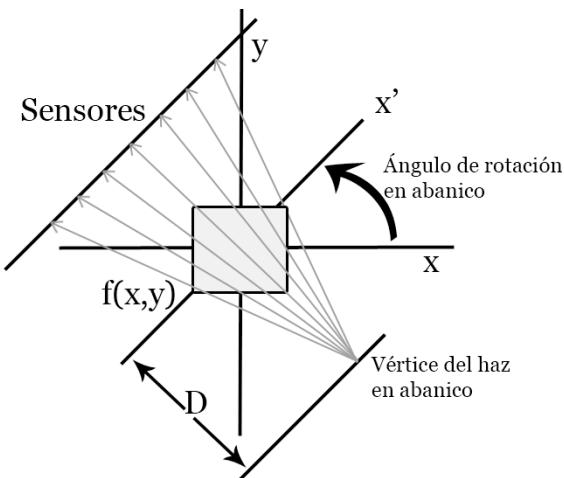
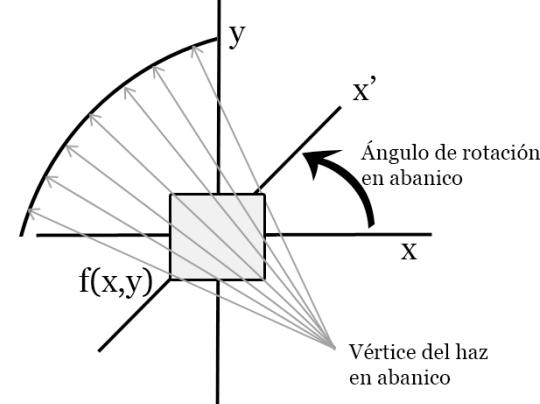


Figura f.162 - Proyecciones Transformada de Radon

Cada columna de `F` contiene las muestras tomadas por el sensor del haz (datos de las proyecciones) a un único ángulo de rotación. El número de columnas en `F` depende del incremento de la rotación en abanico. Por defecto, el incremento de la rotación en abanico es de 1 grado por lo que `F` tiene 360 columnas.

El número de columnas en F se determina a partir del número de sensores. `fanbeam` determina el número de sensores calculando cuántos haces se necesitan para cubrir la imagen entera para cualquier ángulo de rotación.

`F = fanbeam(..., param1, val1, param1, val2,...)` especifica varios parámetros, listados a continuación, que controlan varios aspectos de las proyecciones de haz en abanico. Los nombres de los parámetros se pueden abbreviar y no son sensibles a las mayúsculas:

Parámetro	Descripción
<code>'FanRotationIncrement'</code>	Escalar real positivo que especifica el incremento del ángulo de rotación de las proyecciones de haz en abanico. Medido en grados. (Por defecto: 1)
<code>'FanSensorGeometry'</code>	Cadena que especifica cómo se posicionan los sensores: 'arc' — Los sensores están equiespaciados a través de un arco circular a una distancia D del centro de rotación. (Por defecto)  <i>Figura f.161 - Sensores en arco</i> 'line' — Los sensores están equiespaciados a lo largo de una línea.

	<p>Figura f.162 - Sensores en línea</p>
'FanSensorSpacing'	<p>Escalar real positivo que especifica el espaciado de los sensores de haz en abanico. La interpretación del valor depende de la variable 'FanSensorGeometry' .</p> <p>Si 'FanSensorGeometry' se pone a 'arc' (por defecto) el valor define el espaciado angular en grados (por defecto: 1)</p> <p>Si 'FanSensorGeometry' vale 'line', el valor define el espaciado lineal (por defecto: 1). Consultar la función fanbeam para más detalle.</p> <p>Nota: Este espaciado lineal se mide en el eje x. El origen de los ejes es el pixel central de la imagen.</p>

[F, fan_sensor_positions, fan_rotation_angles] = fanbeam(...) devuelve la localización de los sensores de haz en abanico en la variable fan_sensor_positions y los ángulos de rotación donde se calcularon las proyecciones en la variable fan_rotation_angles. Si el parámetro 'FanSensorGeometry' vale 'arc' (por defecto), la variable fan_sensor_positions contiene los ángulos de medida del sensor de haz en abanico y si 'FanSensorGeometry' vale 'line', la variable contiene las posiciones del sensor a lo largo del eje x'.

I puede ser de tipo numeric o logical. El resto de entradas y salidas tipo numeric pueden ser tipo double. Ninguna de las entradas puede ser de tipo disperso.

Ejemplos de uso

Ejemplo 1: Calcular las proyecciones de haz en abanico de una imagen.

```
% Creamos una imagen de "cabeza de práctica" (phantom head) y calculamos sus proyecciones.
```

```

iptsetpref('ImshowAxesVisible','on')
ph = phantom(512);
imshow(ph)

% Mostramos los ángulos de rotación y las posiciones del sensor.

figure,
imshow(F,[],'XData',Fangles,'YData',Fpos,'InitialMagnification','fit')
axis normal
xlabel('Ángulos de rotación (grados)')
ylabel('Posiciones del sensor (grados)')
colormap(hot), colorbar

```



Figura f.165 - Imagen original (ph)

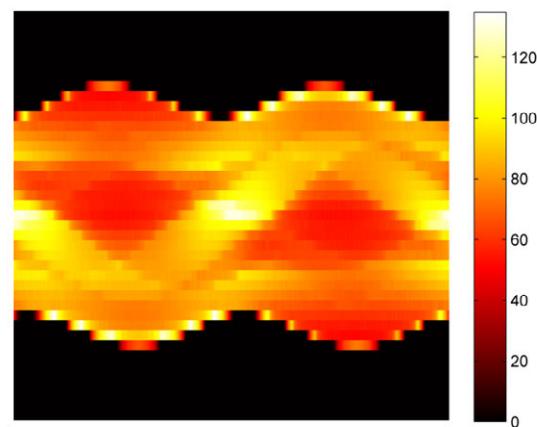


Figura f.166 - Ángulos y posiciones

Ejemplo 2: Comparar las proyecciones de haz en abanico de una imagen para las geometrías de arco y línea con las proyecciones de la Transformada de Radon (en paralelo) para un ángulo de rotación fijo.

```

% Creamos la imagen original, definimos la distancia al vértice del haz y
el incremento de rotación.

I = ones(100);
D = 200;
dtheta = 45;

% Calculamos las proyecciones de haz en abanico para la geometría tipo
% arco y recogemos ángulos de rotación donde se calcularon las
% proyecciones.

[Farc,FposArcDeg,Fangles] =
fanbeam(I,D,'FanSensorGeometry','arc','FanRotationIncrement',dtheta);

% Convertimos posiciones angulares a distancia lineal a lo largo del eje
% x.

FposArc = D*tan(FposArcDeg*pi/180);

```

```
% Calculamos las proyecciones de haz en abanico para la geometría tipo
línea.

[Fline,FposLine] =
fanbeam(I,D,'FanSensorGeometry','line','FanRotationIncrement',dtheta);

% Calculamos la transformada de Radon correspondiente para los ángulos de
rotación de haz en abanico.

[R,Rpos]=radon(I,Fangles);

% Mostramos las tres proyecciones para un ángulo de rotación particular.
Las diferencias se deben a la geometría del muestreado y a las
aproximaciones numéricas en los cálculos.

figure
idx = find(Fangles==45);
plot(Rpos,R(:,idx),FposArc,Farc(:,idx),FposLine,Fline(:,idx))
legend('Radon','Arco','Línea')
```

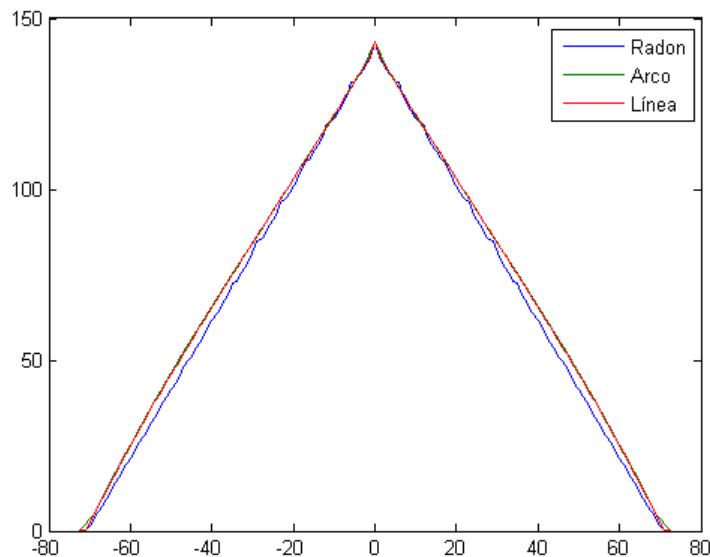


Figura f.167 - Comparativa proyecciones

Funciones relacionadas

`fan2para`, `ifanbeam`, `iradon`, `para2fan`, `phantom`, `radon`

findbounds

Estimación de los límites de salida para una transformación espacial

Sintaxis

```
outbounds = findbounds(TFORM,inbounds)
```

Descripción

outbounds = findbounds(TFORM,inbounds) estima los límites de salida correspondientes a una transformación espacial dada y a un conjunto de límites de entrada. TFORM es una estructura de transformación espacial devuelta por la función maketform. inbounds es una matriz de 2-por-num_dims. La primera fila de inbounds especifica los límites inferiores para cada dimensión y la segunda fila los límites superiores. num_dims tiene que concordar con el campo ndims_in de la variable TFORM.

La función outbounds es del mismo tipo que inbounds. Es una estimación de la región rectangular más pequeña que contiene completamente al rectángulo transformado representado por los límites de entrada especificados. Como outbounds es solo una estimación, puede que no contenga completamente el rectángulo de entrada transformado.

Nota: La función imtransform llama por defecto a findbounds si no se especifican las variables 'XData' e 'YData', parámetros que controlan el recuadro de selección de salida en imtransform. En transformaciones lineales puras, si no se especifican las variables 'XData' e 'YData', la imagen de salida será la misma que la de entrada. Más información en la página de la función imtransform.

Ejemplos de uso

Ejemplo 1: Aplicar una translación lineal entera a una imagen usando findbounds para especificar las variables 'XData' e 'YData' que limitan la salida.

```
% Crear y mostrar imagen original.  
I = checkerboard(50);  
figure, imshow(I)  
  
% Definimos una traslación a lo largo de las componentes horizontal y  
vertical.
```

```
tx = 10;
ty = 6;

% Creamos la estructura de esa transformación.

t = maketform('affine',[1 0 ; 0 1; tx ty]);

% Encontramos los límites de salida para la imagen completa transformada.

bounds = findbounds(t,[1 1; size(I)]);

% Especificamos el recuadro de selección de salida para que empiece donde
% empezaba la imagen original y acabe donde termina la imagen transformada
% y aplicamos la transformación. Sin especificar estos parámetros, la
% imagen final sería igual a la original.

bounds(1,:) = [1 1];
J = imtransform(I,t,'XData',bounds(:,2)', 'YData',bounds(:,1)');

% Mostramos la imagen de salida.

figure, imshow(J)
```

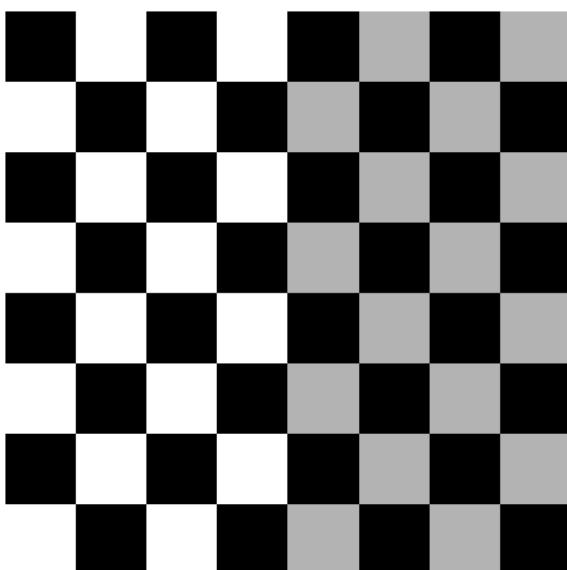


Figura f.168 - Imagen original (I)

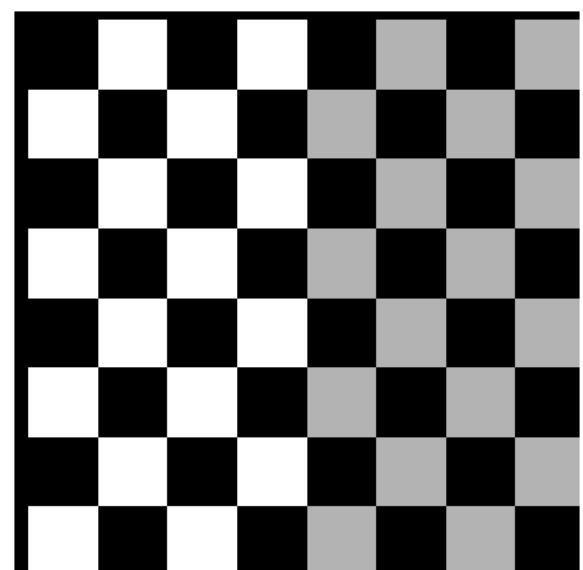


Figura f.169 - Imagen trasladada (J)

Funciones relacionadas

[cp2tfm](#), [imtransform](#), [maketform](#), [tformarray](#), [tformfwd](#), [tforminv](#)

fliptform

Inversión de la función de una estructura TFORM

Sintaxis

```
TFLIP = fliptform(T)
```

Descripción

TFLIP = fliptform(T) crea una nueva estructura de transformación espacial tipo TFORM invirtiendo los roles de las entradas y las salidas en una estructura TFORM existente, de forma que la nueva estructura será una transformación inversa o complementaria a la original.

Ejemplos de uso

Ejemplo 1: Aplicar una transformación a una imagen y rehacer la imagen a con una transformación inversa.

```
I = imread('cameraman.tif');
imshow(I)

% Aplicamos una transformación afín.

th = [.8 .005 0; .1 .9 0; -11 7.4 1];
T = maketform('affine',th);
I2 = imtransform(I, T);
figure, imshow(I2)

% Creamos la transformación inversa.

T2 = fliptform(T)

% Aplicamos la transformación inversa a la imagen original y a la
% transformada.

I3 = imtransform(I, T2);
figure, imshow(I3)

I4 = imtransform(I2, T2);
figure, imshow(I4)
```



Figura f.170 - Imagen original (I)



Figura f.171 - Imagen transformada (I2)



Figura f.172 - Imagen transformada a la inversa (I3)



Figura f.173 - Imagen transformada, transformada a la inversa (I4)

Funciones relacionadas

`maketform`, `tformfwd`, `tforminv`

freqz2

Respuesta en frecuencia en 2-D

Sintaxis

```
[H, f1, f2] = freqz2(h, n1, n2)
[H, f1, f2] = freqz2(h, [n2 n1])
[H, f1, f2] = freqz2(h)
[H, f1, f2] = freqz2(h, f1, f2)
[...] = freqz2(h,...,[dx dy])
[...] = freqz2(h,...,dx)
freqz2(...)
```

Descripción

[H, f1, f2] = freqz2(h, n1, n2) devuelve H, la respuesta en frecuencia de h de tamaño n2-por-n1, y los vectores de frecuencia f1 (de longitud n1) y f2 (de longitud n2). h es un filtro FIR bidimensional en la forma de una molécula computacional. f1 y f2 se devuelven como frecuencias normalizadas en el rango -1.0 a 1.0, donde 1.0 corresponde a la mitad de la frecuencia de muestreo, o π radianes.

[H, f1, f2] = freqz2(h, [n2 n1]) devuelve lo mismo que con [H,f1,f2] = freqz2(h,n1,n2).

[H, f1, f2] = freqz2(h) usa [n2 n1] = [64 64].

[H, f1, f2] = freqz2(h, f1, f2) devuelve la respuesta en frecuencia del filtro FIR h a los valores de frecuencia de f1 y f2. Estos valores de frecuencia deben estar comprendidos en el rango -1.0 a 1.0, donde 1.0 corresponde a la mitad de la frecuencia de muestreo, o π radianes.

[...] = freqz2(h,...,[dx dy]) usa [dx dy] para modificar el espaciado de muestreo en h. dx determina el espaciado para la dimensión x y dy determina el espaciado para la dimensión y. El espaciado por defecto es de 0.5, que corresponde con una frecuencia de muestreo de 2.0.

[...] = freqz2(h,...,dx) usa dx para determinar el espaciado de muestreo en ambas dimensiones.

`freqz2(...)` produce una representación en malla de la magnitud bidimensional de la respuesta en frecuencia cuando no hay especificados parámetros de salida.

La matriz de entrada `h` puede ser del tipo `double` o de cualquier tipo entero. El resto de variables de entrada a `freqz2` deben ser de tipo `double`. Todas las salidas son de tipo `double`.

Ejemplos de uso

Ejemplo 1: Aplicar el método de la ventana para crear un filtro FIR de 16-por-16 y después ver su respuesta en frecuencia usando `freqz2`.

```
% Creamos un filtro FIR mediante el método de la ventana.  
Hd = zeros(16,16);  
Hd(5:12,5:12) = 1;  
Hd(7:10,7:10) = 0;  
h = fwind1(Hd,bartlett(16));  
  
colormap(jet(64))  
  
% Obtenemos la respuesta en frecuencia de 32-por-32 de filtro FIR y los  
vectores de frecuencia f1 (de longitud 32) y f2 (de longitud 32).  
  
axis([-1 1 -1 1 0 1]);  
  
freqz2(h,[32 32]);  
  
% Aumentamos el tamaño para verlo más sólido.  
figure, freqz2(h,[500 500]);
```

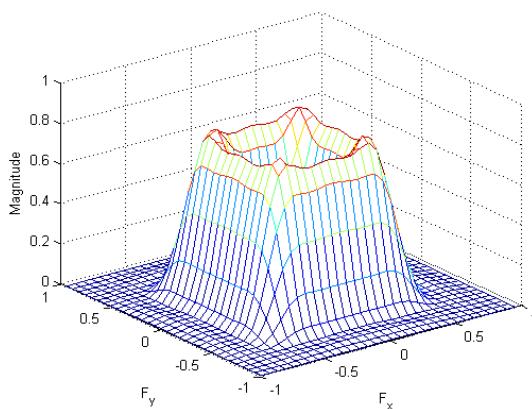


Figura f.174 - Respuesta en frecuencia 32x32

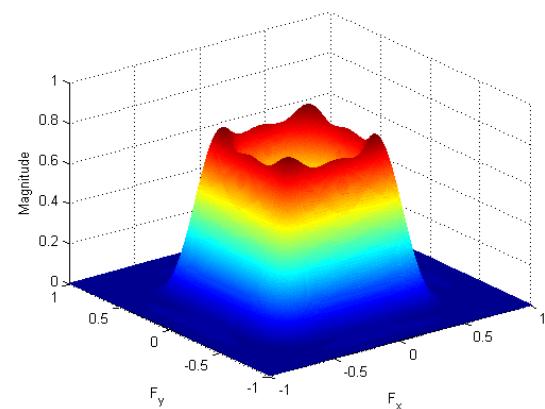


Figura f.175 - Respuesta en frecuencia 500x500

Ejemplo 2: Crear la respuesta en frecuencia de un filtro en 2D predefinido

```
% Por ejemplo, un filtro Gaussiano paso bajo.
```

```
h = fspecial('gaussian');  
freqz2(h, 500, 500);
```

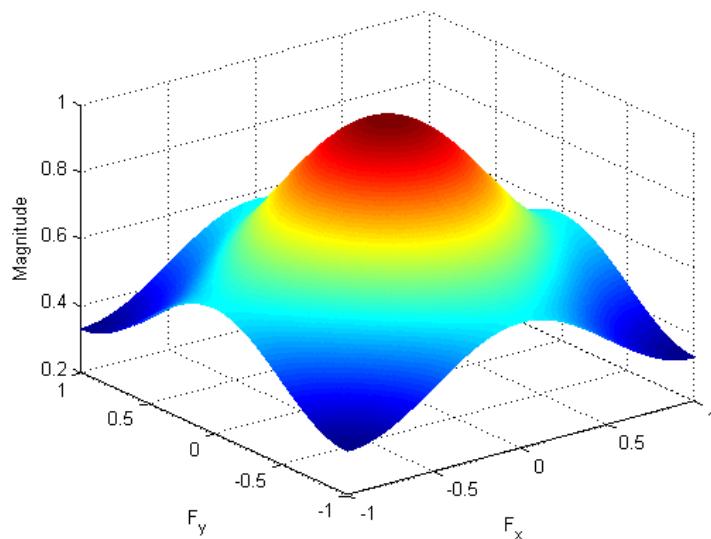


Figura f.176 - Respuesta en frecuencia filtro gaussiano

Funciones relacionadas

`freqz`

fsamp2

Creación de un filtro FIR en 2-D utilizando muestreo en frecuencia

Sintaxis

```
h = fsamp2(Hd)
h = fsamp2(f1, f2, Hd, [m n])
```

Descripción

`h = fsamp2(Hd)` diseña un filtro FIR bidimensional a partir de su respuesta en frecuencia `Hd` y devuelve los coeficientes del filtro en una matriz `h`. `fsamp2` devuelve `h` como una molécula computacional, que es la forma adecuada para trabajar con la función `filter2`. El filtro `h` tiene una respuesta en frecuencia que va a través de los puntos definidos en `Hd`. Si `Hd` es de tamaño `m`-por-`n`, entonces `h` será también de `m`-por-`n`.

`fsamp2` diseña un filtro FIR bidimensional basándose en la respuesta en frecuencia deseada muestreada en puntos del plano cartesiano. `Hd` es una matriz que contiene la respuesta en frecuencia deseada muestreada en puntos equiespaciados entre -1.0 y 1.0 a lo largo de los ejes de frecuencia `x` e `y`, donde 1.0 corresponde a la mitad de la frecuencia de muestreo o π radianes. Para obtener resultados más exactos, utilizar puntos de frecuencia devueltos por la función `freqspace` para crear `Hd`.

`h = fsamp2(f1, f2, Hd, [m n])` produce un filtro FIR de `m`-por-`n` haciendo coincidir la respuesta del filtro `Hd` con los puntos definidos con los vectores `f1` y `f2`. Los vectores de frecuencia `f1` y `f2` están normalizados en frecuencia, donde 1.0 se corresponde a la mitad de la frecuencia de muestreo o π radianes. El filtro resultante encaja en la respuesta en frecuencia deseada lo máximo posible según el criterio de mínimo error cuadrático. Para obtener los mejores resultados, debe haber al menos `m*n` puntos de frecuencias deseadas. `fsamp2` dará un mensaje de aviso si se especifican menos puntos.

La matriz de entrada `Hd` puede ser de tipo `double` o de cualquier tipo entero. El resto de entradas deben ser de tipo `double`. Todas las salidas son de tipo `double`.

Ejemplos de uso

Ejemplo 1: Utilizar `fsamp2` para diseñar un filtro pasobanda aproximadamente simétrico con banda de paso entre 0.1 y 0.5 (frecuencia normalizada donde 1.0 corresponde a la mitad de la frecuencia de muestreo o π radianes):

```
% Creamos una matriz Hd que contiene la respuesta pasobanda deseada. Para
ellos usamos la función freqspace que creará los vectores de rango de
frecuencia f1 y f2.
```

```
[f1,f2] = freqspace(21,'meshgrid');
```

```
Hd = ones(21);
```

```
r = sqrt(f1.^2 + f2.^2);
```

```
Hd((r<0.1)|(r>0.5)) = 0;
```

```
% Mostramos la respuesta en frecuencia.
```

```
colormap(jet(64))
```

```
mesh(f1,f2,Hd)
```

```
% Creamos el filtro a partir de la respuesta en frecuencia y lo
mostramos.
```

```
h = fsamp2(Hd);
```

```
figure, freqz2(h)
```

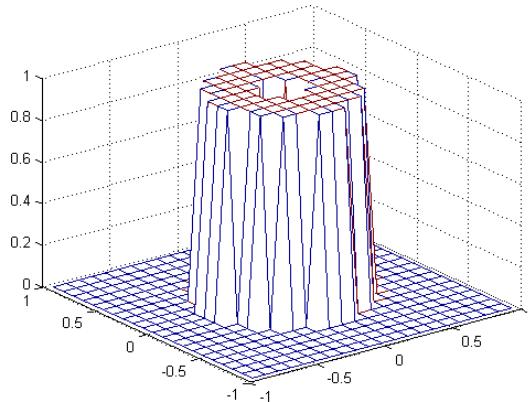


Figura f.177 - Respuesta en frecuencia pasobanda

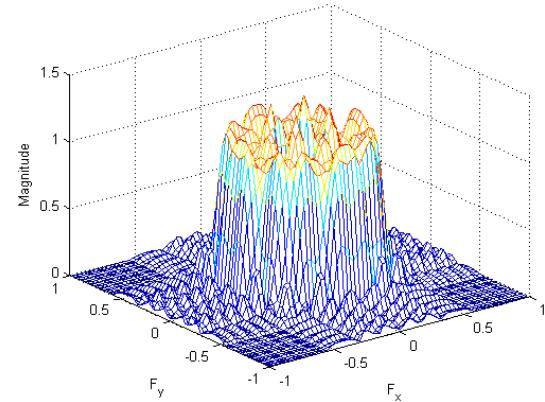


Figura f.178 - Filtro con respuesta en frecuencia
pasobanda

Funciones relacionadas

`conv2`, `filter2`, `freqspace`, `ftrans2`, `fwind1`, `fwind2`

fspecial

Creación de un filtro predefinido en 2-D

Sintaxis

```
h = fspecial(type)
h = fspecial(type, parameters)
```

Descripción

`h = fspecial(type)` crea un filtro bidimensional `h` del tipo especificado en la variable `type`. `fspecial` devuelve `h` como un kernel de correlación, que es la forma apropiada para usar con la función `imfilter`. `type` es una cadena que puede ser cualquiera de los siguientes valores:

Valor	Descripción
'average'	Filtro promedio, útil para eliminar ruido y desenfocar.
'disk'	Filtro promedio circular (<i>pillbox</i>).
'gaussian'	Filtro Gaussiano pasobajo.
'laplacian'	Filtro Laplaciano, útil para resaltar bordes.
'log'	Laplaciano del Gaussiano, útil para resaltar bordes.
'motion'	Filtro para aproximar el movimiento lineal de una cámara.
'prewitt'	Filtro de Prewitt enfatizador de bordes.
'sobel'	Filtro de Sobel enfatizador de bordes.
'unsharp'	Filtro <i>Unsharp</i> para mejorar el enfoque.

`h = fspecial(type, parameters)` usa el filtro especificado en la variable `type` además de varios parámetros adicionales especificados en la variable `parameters` que son específicos del tipo de filtro elegido. Si se omiten estos parámetros se usarán los valores por defecto. A continuación se muestra una lista con la sintaxis y parámetros específicos para cada tipo de filtro:

- `h = fspecial('average', hsize)` devuelve un filtro promedio de tamaño `hsize` en la variable `h`. El argumento `hsize` puede ser un vector que especifique el número de filas y columnas en `h`, o puede ser una escalar, en cuyo caso `h` será una matriz cuadrada. El valor por defecto para `hsize` es [3 3].
- `h = fspecial('disk', radius)` devuelve un filtro promedio circular (*pillbox*) dentro de la matriz cuadrada de lado `2*radius+1`. El valor por defecto para `radius` es 5.
- `h = fspecial('gaussian', hsize, sigma)` devuelve un filtro Gaussiano pasabajo rotacionalmente simétrico de tamaño `hsize` con una desviación estándar de `sigma` (positiva). `hsize` puede ser un vector que especifique el número de filas y columnas de `h`, o puede ser una escalar, en cuyo caso `h` será una matriz cuadrada. El valor por defecto para `hsize` es [3 3] y 0.5 para `sigma`.
- `h = fspecial('laplacian', alpha)` devuelve un filtro de 3-por-3 aproximando la forma del operador Laplaciano bidimensional. El parámetro `alpha` controla la forma del Laplaciano y debe estar en el rango de 0.0 a 1.0. El valor por defecto para `alpha` es 0.2.
- `h = fspecial('log', hsize, sigma)` devuelve un filtro Laplaciano del Gaussiano rotacionalmente simétrico, de tamaño `hsize` y con una desviación estándar `sigma` (positiva). `hsize` puede ser un vector que especifique el número de filas y columnas de `h`, o puede ser una escalar, en cuyo caso `h` será una matriz cuadrada. El valor por defecto para `hsize` es [5 5] y 0.5 para `sigma`.
- `h = fspecial('motion', len, theta)` devuelve un filtro para aproximarse, una vez convolucionado con una imagen, al movimiento lineal de una cámara de `len` píxeles, con un ángulo de `theta` grados en dirección contraria a las agujas del reloj. El filtro se convierte en un vector para los movimientos horizontales y verticales. El valor por defecto para `len` es 9 y para `theta` es 0, que corresponde con un movimiento horizontal de nueve píxeles. Para calcular los coeficientes `h` del filtro se debe proceder de la siguiente manera:
 1. Construir un segmento recto ideal con la longitud y ángulos deseados, centrado en el coeficiente central de `h`.
 2. Para cada localización de coeficiente (`i, j`), calcular la distancia más cercana (`nearest_distance`) entre esa localización y el segmento.
 3. `h = max(1 - nearest_distance, 0);`

4. Normalizar h: $h = h / (\text{sum}(h(:)))$

- $h = \text{fspecial}('prewitt')$ devuelve un filtro h de 3-por-3 (mostrado debajo) que enfatiza los bordes horizontales mediante la aproximación a un gradiente vertical. Si se quieren potenciar los bordes verticales se debe realizar la traspuesta del filtro (h^*) .

```
[1 1 1  
 0 0 0  
 -1 -1 -1]
```

- $h = \text{fspecial}('sobel')$ devuelve un filtro h de 3-por-3 (mostrado debajo) que enfatiza los bordes horizontales utilizando el efecto de suavizado logrado mediante la aproximación a un gradiente vertical. Si se quieren potenciar los bordes verticales se debe realizar la traspuesta del filtro (h^*) .

```
[1 2 1  
 0 0 0  
 -1 -2 -1]
```

- $h = \text{fspecial}('unsharp', alpha)$ devuelve un filtro *unsharp* de mejora del enfoque de tamaño 3-por-3. *fspecial* crea el filtro a partir del negativo del filtro Laplaciano usando el parámetro *alpha*, parámetro que controla la forma del Laplaciano y debe estar en el rango de 0.0 a 1.0. El valor por defecto para *alpha* es 0.2.

El nombre del filtro procede de un proceso en el que se enfoca una imagen a partir de sustraer una imagen borrosa (*unsharp*) de la propia imagen.

h es de clase *double*.

Ejemplos de uso

Ejemplo 1: Aplicar todos los filtros a una imagen y mostrar cada resultado por separado.

```
I = imread('cameraman.tif');  
imshow(I)  
  
H = fspecial('average',[10 10])  
Average = imfilter(I,H,'replicate');  
figure, imshow(Average)  
  
H = fspecial('disk',10);  
blurred = imfilter(I,H,'replicate');  
figure, imshow(blurred)
```

```
H = fspecial('gaussian',[20 20],10);
Gaussian = imfilter(I,H,'replicate');
figure, imshow(Gaussian)

H = fspecial('laplacian');
Laplacian = imfilter(I,H,'replicate');
figure, imshow(Laplacian)

H = fspecial('log');
Log = imfilter(I,H,'replicate');
figure, imshow(Log)

H = fspecial('motion',20);
MotionBlur = imfilter(I,H,'replicate');
figure, imshow(MotionBlur)

H = fspecial('prewitt');
Prewitt = imfilter(I,H,'replicate');
figure, imshow(Prewitt)

H = fspecial('sobel');
Sobel = imfilter(I,H,'replicate');
figure, imshow(Sobel)

H = fspecial('unsharp',0.5);
sharpened = imfilter(I,H,'replicate');
figure, imshow(sharpened)
```

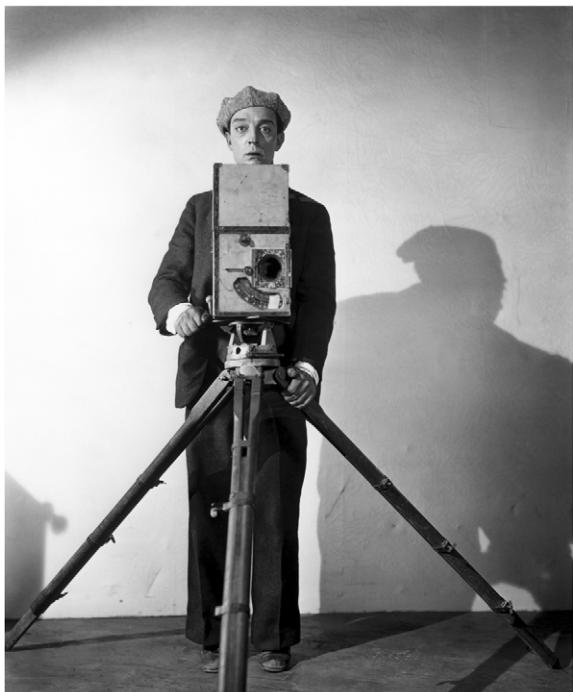


Figura f.179 - Imagen original (I)



Figura f.180 - Filtrado promedio (Average)



Figura f.181 - Filtrado disco (blurred)



Figura f.182 - Filtrado Gaussiano (Gaussian)

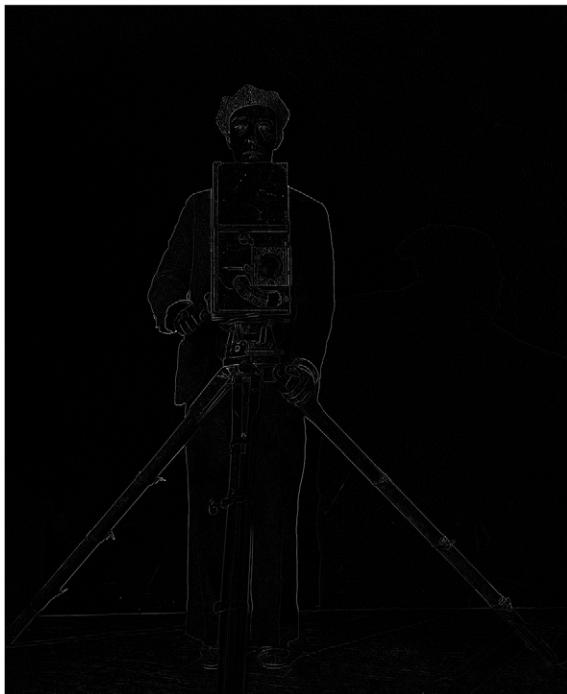


Figura f.183 - Filtrado Laplaciano (Laplacian)

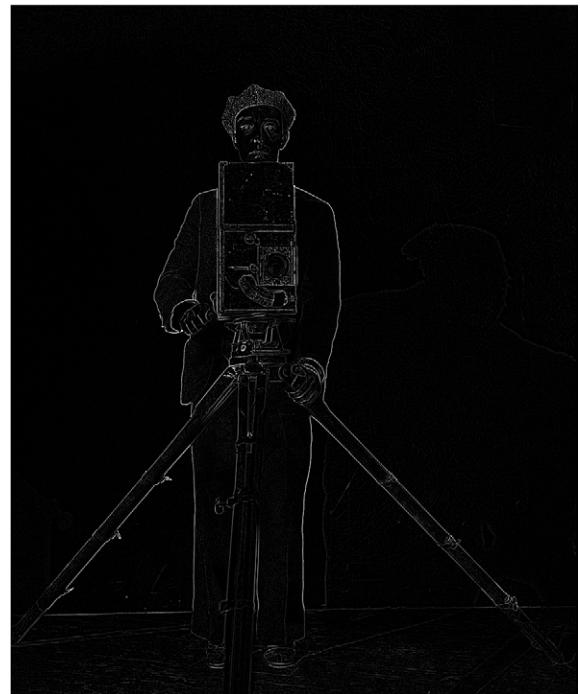


Figura f.184 - Filtrado Laplaciano del Gaussiano
(Log)



Figura f.185 - Filtrado movimiento lineal
(MotionBlur)

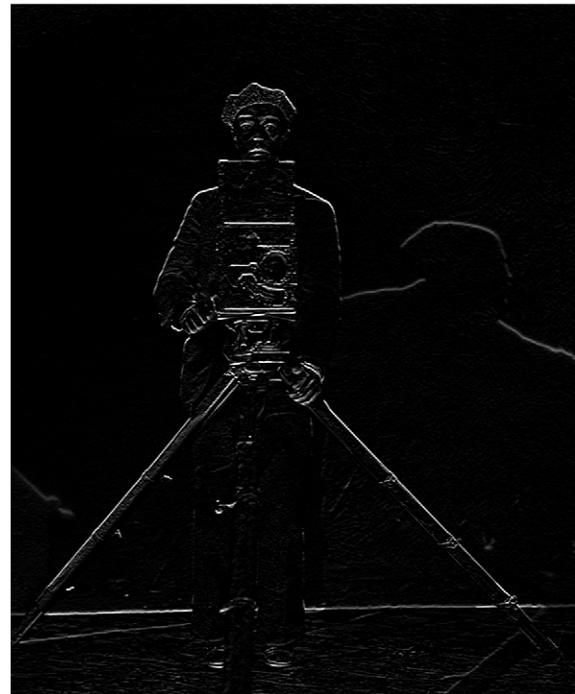


Figura f.186 - Filtrado Prewitt (Prewitt)

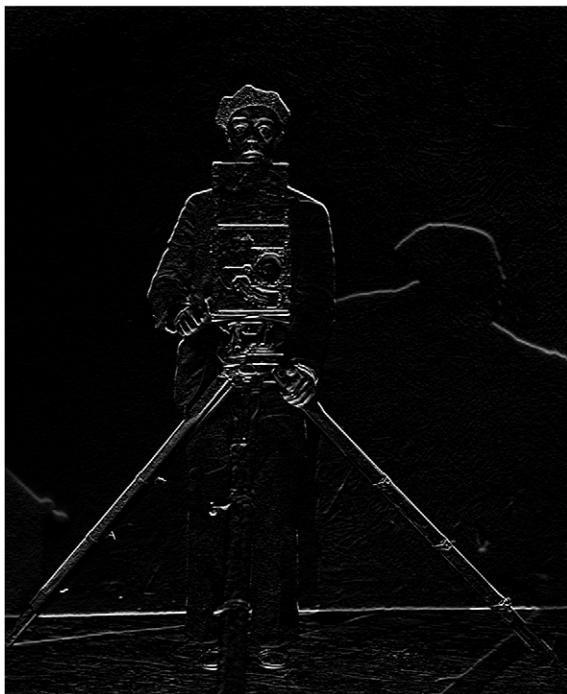


Figura f.187 - Filtrado Sobel (Sobel)



Figura f.188 - Filtrado de enfoque (sharpened)

Funciones relacionadas

`conv2`, `edge`, `filter2`, `fsamp2`, `fwindl`, `fwind2`, `imfilter`, `del2`

ftrans2

Creación de un filtro FIR en 2-D a partir de uno en 1-D utilizando una transformación de frecuencia

Sintaxis

```
h = ftrans2(b, t)
h = ftrans2(b)
```

Descripción

`h = ftrans2(b, t)` produce el filtro FIR bidimensional `h` que se corresponde con el filtro FIR de una dimensión `b` al que se le ha aplicado la transformación `t`. `ftrans2` devuelve `h` como una molécula computacional, que es la forma adecuada para trabajar con la función `filter2`. `b` debe ser un filtro de una dimensión, Tipo I (simetría par, número de coeficientes impar) como los filtros devueltos por las funciones `fir1`, `fir2`, o `remez` de la *toolbox*. La matriz de transformación `t` contiene los coeficientes que definen la transformación de frecuencia que se usará. Si `t` es de tamaño `m`-por-`n` y `b` tiene longitud `Q`, entonces `h` será de tamaño $((m-1)*(Q-1)/2+1)$ -por- $((n-1)*(Q-1)/2+1)$.

`h = ftrans2(b)` usa por defecto la matriz de transformación de McClellan `t`.

```
t = [1 2 1; 2 -4 2; 1 2 1]/8;
```

Todas las entradas y salidas deben de ser de tipo `double`.

Ejemplos de uso

Ejemplo 1: Utilizar `ftrans2` para diseñar un filtro pasobanda bidimensional con simétrica esférica aproximada y una banda de paso entre 0.1 y 0.6 (frecuencia normalizada donde 1.0 corresponde a la mitad de la frecuencia de muestreo o π radianes):

```
% Diseñamos y mostramos primero el filtro pasobanda FIR de una dimensión
% al que luego pasaremos a dos dimensiones.
```

```
colormap(jet(64))
b = remez(10,[0 0.05 0.15 0.55 0.65 1],[0 0 1 1 0 0]);
[H,w] = freqz(b,1,128,'whole');
plot(w/pi-1,fftshift(abs(H)))
```

% Usamos ftrans2 con la transformación McClellan por defecto para convertir el filtro a 2-D y obtener un filtro con simétrica esférica aproximada como queríamos.

```
h = ftrans2(b);
figure, freqz2(h)
```

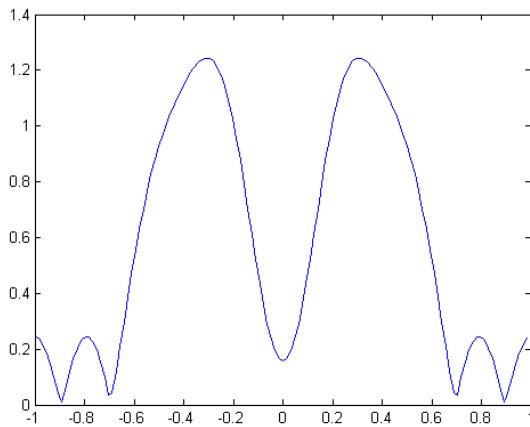


Figura f.189 - Respuesta en frecuencia FIR pasobanda 1-D

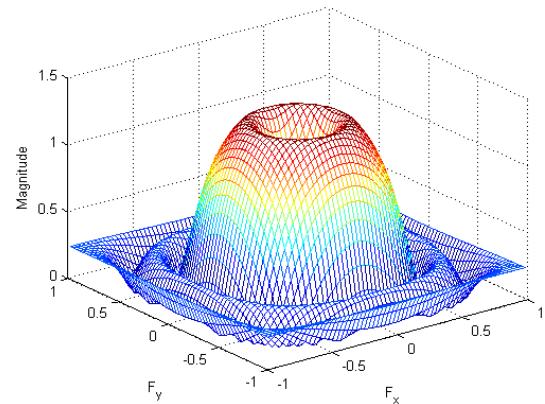


Figura f.190 - Filtro FIR 2-D con respuesta en frecuencia pasobanda

Funciones relacionadas

conv2, filter2, fsamp2, fwind1, fwind2

fwind1

Creación de un filtro FIR en 2-D utilizando el método de la ventana

Sintaxis

```
h = fwind1(Hd, win)
h = fwind1(Hd, win1, win2)
h = fwind1(f1, f2, Hd,...)
```

Descripción

fwind1 diseña filtros FIR bidimensionales utilizando el método de la ventana. fwind1 usa las especificaciones de una ventana unidimensional para diseñar el filtro FIR bidimensional basándose en la respuesta en frecuencia deseada H_d . fwind1 solo trabaja con ventanas unidimensionales y para trabajar con ventanas bidimensionales se debe usar la función fwind2.

$h = \text{fwind1}(H_d, \text{win})$ diseña un filtro FIR bidimensional h con la respuesta en frecuencia especificada en la variable H_d . fwind1 devuelve h como una molécula computacional, que es la forma adecuada para trabajar con la función filter2. fwind1 usa una ventana unidimensional para crear una ventana bidimensional con simétrica esférica aproximada utilizando el método de Huang. Se puede especificar la ventana en la variable win utilizando funciones de la *toolbox* de Procesado de Señales como boxcar, hamming, hanning, bartlett, blackman, kaiser, o chebwin. Si $\text{length}(\text{win})$ es n , entonces h será de tamaño n -por- n .

H_d es una matriz que contiene la respuesta en frecuencia deseada, muestreada en puntos equiespaciados entre -1.0 y 1.0 a lo largo de los ejes de frecuencia x e y , donde 1.0 corresponde a la mitad de la frecuencia de muestreo o π radianes. Para obtener resultados más exactos, crear H_d utilizando puntos de frecuencia devueltos por la función freqspace.

$h = \text{fwind1}(H_d, \text{win1}, \text{win2})$ usa la ventana bidimensional especificada en las variables win1 y win2 para crear una ventana bidimensional separable. Si $\text{length}(\text{win1})$ es n y $\text{length}(\text{win2})$ es m , entonces h será de tamaño m -por- n .

$h = \text{fwind1}(f1, f2, H_d, \dots)$ permite especificar la respuesta en frecuencia deseada H_d en frecuencias arbitrarias ($f1$ y $f2$) a lo largo de los ejes x e y . Los vectores de frecuencia

f_1 y f_2 deben estar comprendidos en el rango -1.0 a 1.0, frecuencia normalizada donde 1.0 corresponde a la mitad de la frecuencia de muestreo o π radianes. La longitud de las ventanas controla el tamaño del filtro resultante, igual que en la definición anterior.

La matriz de entrada H_d puede ser de tipo `double` o de cualquier tipo entero. El resto de entradas a `fwind1` deben ser de tipo `double`. Todas las salidas son de tipo `double`.

Ejemplos de uso

Ejemplo 1: Utilizar `fwind1` para diseñar un filtro pasobanda bidimensional con simétrica esférica aproximada y banda de paso entre 0.1 y 0.5 (frecuencia normalizada donde 1.0 corresponde a la mitad de la frecuencia de muestreo o π radianes):

```
% Creamos y mostramos la matriz Hd que contiene la respuesta pasobanda deseada. Usamos la función freqspace para crear los vectores de rango de frecuencia f1 y f2.
```

```
[f1,f2] = freqspace(21,'meshgrid');
Hd = ones(21);
r = sqrt(f1.^2 + f2.^2);
Hd((r<0.1)|(r>0.5)) = 0;
colormap(jet(64))
mesh(f1,f2,Hd)
```

```
% Diseñamos el filtro usando una ventana Hamming unidimensional.
```

```
h = fwind1(Hd,hamming(21));
figure, freqz2(h)
```

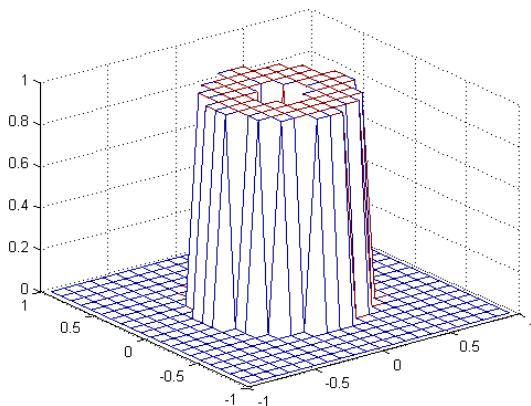


Figura f.191 - Respuesta pasobanda 1-D

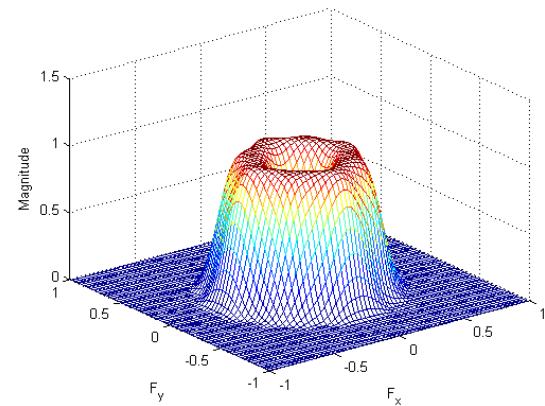


Figura f.192 - Filtro con respuesta pasobanda 2-D

Funciones relacionadas

`conv2`, `filter2`, `fsamp2`, `freqspace`, `ftrans2`, `fwind2`

fwind2

Creación de un filtro FIR en 2-D utilizando el método de la ventana en 2-D

Sintaxis

```
h = fwind2(Hd, win)
h = fwind2(f1, f2, Hd, win)
```

Descripción

Fwind2 diseña filtros FIR bidimensionales utilizando el método de la ventana bidimensional. Fwind2 usa las especificaciones de una ventana bidimensional para diseñar el filtro FIR bidimensional basándose en la respuesta en frecuencia deseada H_d . Fwind2 solo trabaja con ventanas bidimensionales y para trabajar con ventanas unidimensionales se debe usar la función fwind1.

$h = \text{fwind2}(H_d, \text{win})$ crea el filtro FIR bidimensional h utilizando una transformada inversa de Fourier de la respuesta en frecuencia deseada H_d y la multiplicación por la ventana win . H_d es una matriz que contiene la respuesta en frecuencia muestreada en puntos equiespaciados en el plano Cartesiano. fwind2 devuelve h como una molécula computacional, que es la forma adecuada para trabajar con la función filter2. h es del mismo tamaño que win . Para obtener resultados más exactos, crear H_d utilizando puntos de frecuencia devueltos por la función freqspace.

$h = \text{fwind2}(f_1, f_2, H_d, \text{win})$ permite especificar la respuesta en frecuencia deseada H_d en frecuencias arbitrarias (f_1 y f_2) a lo largo de los ejes x e y . Los vectores de frecuencia f_1 y f_2 deben estar comprendidos en el rango -1.0 a 1.0, frecuencia normalizada donde 1.0 corresponde a la mitad de la frecuencia de muestreo o π radianes. h es del mismo tamaño que win .

La matriz de entrada H_d puede ser de tipo double o de cualquier tipo entero. El resto de entradas a fwind2 deben ser de tipo double. Todas las salidas son de tipo double.

Ejemplos de uso

Ejemplo 1: Utilizar `fwind2` para diseñar un filtro pasobanda bidimensional con simétrica esférica aproximada y banda de paso entre 0.1 y 0.5 (frecuencia normalizada donde 1.0 corresponde a la mitad de la frecuencia de muestreo o π radianes):

```
% Creamos y mostramos la matriz Hd que contiene la respuesta pasobanda deseada. Usamos la función freqspace para crear los vectores de rango de frecuencia f1 y f2.
```

```
[f1,f2] = freqspace(21,'meshgrid');
Hd = ones(21);
r = sqrt(f1.^2 + f2.^2);
Hd((r<0.1)|(r>0.5)) = 0;
colormap(jet(64))
mesh(f1,f2,Hd)
```

```
% Creamos una ventana Gaussiana bidimensional con la función fspecial, la mostramos y diseñamos y mostramos el filtro con la ventana.
```

```
win = fspecial('gaussian',21,2);
win = win ./ max(win(:)); % Normalizamos a 1 el valor máximo de ventana.
figure, mesh(win)
h = fwind2(Hd,win);
figure, freqz2(h)
```

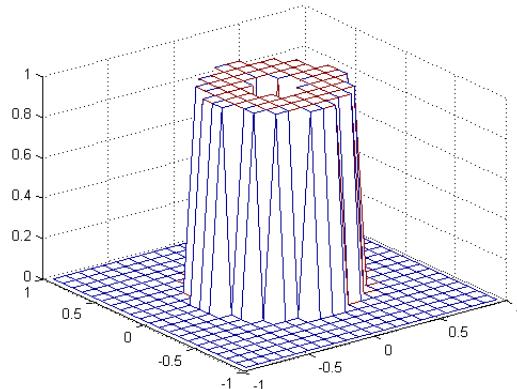


Figura f.193 - Respuesta pasobanda 1-D

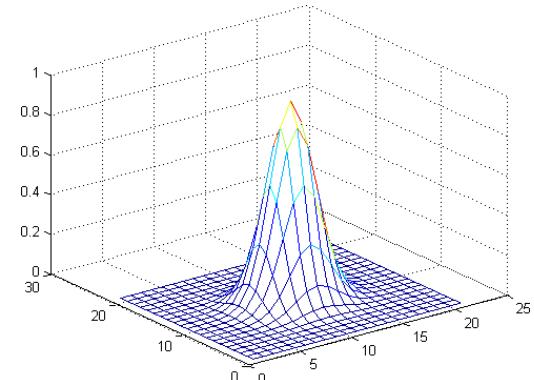


Figura f.194 - Ventana Gaussiana 2-D

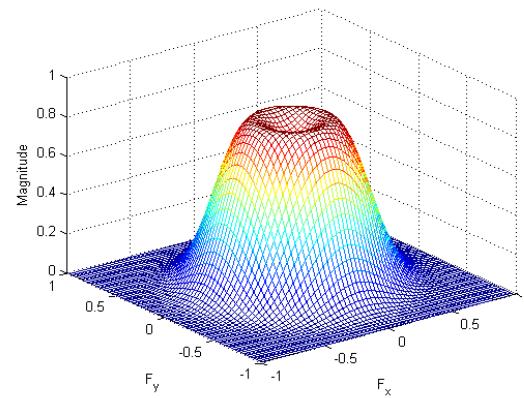


Figura f.195 - Filtro con respuesta pasobanda 2-D

Funciones relacionadas

`conv2`, `filter2`, `fsamp2`, `freqspace`, `ftrans2`, `fwind1`

getheight

Obtención de la altura de un elemento estructurante

Introducción

Los elementos estructurantes son un tipo de objeto definido en MATLAB esencial en las operaciones morfológicas para examinar la imagen de entrada ya que sirven como patrón de búsqueda. Se utiliza la función `strel` para crearlos.

Los elementos estructurantes bidimensionales o planos consisten fundamentalmente en una matriz de 0's y 1's y son normalmente mucho menores que la imagen a procesar. En los elementos estructurantes tridimensionales o no planos los 0's y 1's definen la extensión del elemento estructurante en el plano-x y en el plano-y, y utilizan además valores de altura asociados con cada pixel para definir la tercera dimensión. El pixel central del elemento estructurante, al que se le llama origen, identifica el pixel de interés, el pixel que se está procesando. Los píxeles con valor 1 definen la vecindad del elemento estructurante y se les denomina vecinos del elemento estructurante. La localización de los vecinos se hace referente al pixel central con la notación [vertical, horizontal] que indica el desplazamiento vertical y horizontal del vecino respecto el origen:

1	0	0
1	0	0
1	0	1

Vecindad =
[1 0 0; 1 0 0; 1 0 1];
Localización =
[-1 -1; 0 -1; 1 -1; 1 1];

Figura f.196 - Elemento estructurante con vecindad y localizaciones

Sintaxis

```
H = getheight(SE)
```

Descripción

`H = getheight(SE)` devuelve un array del mismo tamaño que el array devuelto por `getnhood(SE)` con los valores de la altura asociada a cada vecino del elemento estructurante `SE`. `H` es todo ceros para un elemento estructurante bidimensional o plano, ya que no tienen una altura que defina su tercera dimensión. `SE` es un elemento estructurante como el devuelto por la función `strel`. `H` es de clase `double`.

Al crear un elemento estructurante con la función `strel` sin deshabilitar la salida de datos por pantalla mediante el operador `:` (punto y coma) siempre se muestra tanto la vecindad como la altura del elemento estructurante creado.

Ejemplos de uso

Ejemplo 1: Crear un elemento estructurante tridimensional especificando su altura y leer luego su altura en una variable.

```
se = strel(ones(3,3),magic(3));
getheight(se)

se =
Nonflat STREL object containing 9 neighbors.

Neighborhood:
    1     1     1
    1     1     1
    1     1     1

Height:
    8     1     6
    3     5     7
    4     9     2

ans =
```

8	1	6
3	5	7
4	9	2

Funciones relacionadas

`strel`, `getnhood`

getimage

Lectura de los datos de una imagen mostrada en los ejes

Sintaxis

```
A = getimage(h)
[x, y, A] = getimage(h)
[..., A, flag] = getimage(h)
[...] = getimage
```

Descripción

`A = getimage(h)` devuelve los primeros datos de imagen que haya en el identificador de objeto gráfico `h`. `h` puede ser una figura, eje, o una imagen. `A` es idéntico a la propiedad de la imagen `CData` que guarda el color de cada área; contiene los mismos valores y es del mismo tipo (`uint8`, `uint16`, `double` o `logical`). Si `h` no es una imagen o no contiene una imagen, `A` volverá vacío.

`[x, y, A] = getimage(h)` devuelve las propiedades de la imagen `XData` en la variable `x` e `YData` en `y`. `XData` e `YData` son vectores de dos elementos que indican el rango del eje `x` e `y` respectivamente.

`[..., A, flag] = getimage(h)` devuelve `flag`, un indicador entero que marca el tipo de imagen contenida en `h`. La siguiente tabla resume sus posibles valores.

Valor	Tipo de Imagen
0	No es una imagen; <code>A</code> se devuelve como matriz vacía.
1	Imagen indexada.
2	Imagen de intensidad con valores comprendidos en el rango estándar ([0, 1] para arrays tipo <code>single</code> y <code>double</code> , [0, 255] para arrays <code>uint8</code> , [0, 65535] para arrays <code>uint16</code>).
3	Datos de intensidad, pero no en el rango estándar.
4	Imagen RGB.
5	Imagen binaria.

[. . .] = getimage devuelve información del objeto de ejes actual. Es equivalente a
[. . .] = getimage(gca).

El array de salida A es del mismo tipo que la propiedad de la imagen CData. El resto de entradas y salidas son de tipo double.

Nota: Para imágenes tipo int16 y single, los datos de imagen devueltos por la función getimage son de tipo double, no int16 o single, porque la función lee los datos de la propiedad CData que como el resto de objetos de imagen, guarda este tipo de datos como tipo double.

Por ejemplo, al crear un objeto de imagen de tipo int16, si se lee la propiedad CData del objeto y se mira su clase, será tipo double.

```
h = imshow(ones(10,'int16'));
class(get(h,'CData'))
```

Así que si se obtiene este dato usando la función getimage, el tipo de dato será double y la variable flag con el indicador de tipo será igual a 3.

```
[img,flag] = getimage(h);
class(img)
```

Para imágenes tipo single, el valor de flag será 2 porque los tipos single y double tienen el mismo rango dinámico.

Ejemplos de uso

Ejemplo 1: Después de utilizar la función imshow o imtool para mostrar una imagen directamente de un archivo, usar getimage para incorporar todos los datos de la imagen al espacio de trabajo.

```
imshow rice.png
I = getimage;
```

Funciones relacionadas

imshow, imtool

getimagemodel

Creación de un objeto ‘modelo de imagen’ desde un objeto **imagen**

Introducción

Un objeto modelo de imagen guarda cierta información de una imagen: clase, tipo, rango de intensidades, anchura, altura, mínimo y máximo valor de intensidad. El objeto acepta métodos que se pueden usar para acceder y mostrar esta información. Para saber la lista de métodos aceptados se debe escribir `methods imagemode1` o `help imagemode1/método` para más información a cerca de un método determinado.

Sintaxis

```
imgmodel = getimagemode1(himage)
```

Descripción

`imgmodel = getimagemode1(himage)` devuelve el objeto modelo de imagen asociado con `himage` o crea uno nuevo si no tiene asignado uno. `himage` debe ser un puntero a un objeto de imagen o un array de punteros a objetos de imagen, en cuyo caso `imgmodel` será un array de modelos de imagen.

Ejemplos de uso

Ejemplo 1: Crear un objeto modelo de imagen a partir de una imagen y mostrarlo.

```
h = imshow('bag.png')
imgmodel = getimagemode1(h)

imgmodel =
    IMAGEMODEL object accessing an image with these properties:
        ClassType: 'uint8'
        DisplayRange: [0 255]
        ImageHeight: 250
        ImageType: 'intensity'
        ImageWidth: 189
        MinIntensity: 0
        MaxIntensity: 255
```

Funciones relacionadas

`imagemode1`

getline

Obtención de las coordenadas de una polilínea mediante su selección en una figura

Introducción

Una polilínea es un objeto formado por uno o más segmentos de línea (rectas o arcos).

Sintaxis

```
[x, y] = getline(fig)
[x, y] = getline(ax)
[x, y] = getline
[x, y] = getline(...,'closed')
```

Descripción

`[x, y] = getline(fig)` permite seleccionar interactivamente una polilínea en los ejes actuales de la figura `fig` utilizando el ratón y recoger sus coordenadas en las variables `x` e `y`. Se debe usar un clic derecho del ratón para añadir puntos a la polilínea. Teniendo presionadas las teclas Shift o derecha al hacer clic, o haciendo doble clic, se añade un punto final y se finaliza la selección de la polilínea. Presionando Return o Enter se finaliza la selección sin añadir un punto final. Presionando Backspace o Delete se elimina de la polilínea el punto seleccionado.

`[x, y] = getline(ax)` permite seleccionar una polilínea en los ejes especificados por el puntero `ax`.

`[x, y] = getline` es lo mismo que `[x,y] = getline(gcf)`.

`[x, y] = getline(...,'closed')` permite seleccionar un polígono cerrado.

Ejemplos de uso

Ejemplo 1: Crear una polilínea en una imagen y recoger sus coordenadas.

```
imshow('moon.tif')
[x,y] = getline
```

Funciones relacionadas

`getpts`, `getrect`

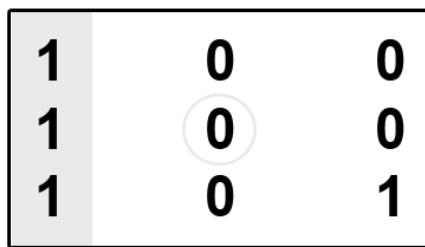
getneighbors

Obtención de las localizaciones y alturas de los vecinos de un elemento estructurante

Introducción

Los elementos estructurantes son un tipo de objeto definido en MATLAB esencial en las operaciones morfológicas para examinar la imagen de entrada ya que sirven como patrón de búsqueda. Se utiliza la función `strel` para crearlos.

Los elementos estructurantes bidimensionales o planos consisten fundamentalmente en una matriz de 0's y 1's y son normalmente mucho menores que la imagen a procesar. En los elementos estructurantes tridimensionales o no planos los 0's y 1's definen la extensión del elemento estructurante en el plano-x y en el plano-y, y utilizan además valores de altura asociados con cada pixel para definir la tercera dimensión. El pixel central del elemento estructurante, al que se le llama origen, identifica el pixel de interés, el pixel que se está procesando. Los píxeles con valor 1 definen la vecindad del elemento estructurante y se les denomina vecinos del elemento estructurante. La localización de los vecinos se hace referente al pixel central con la notación [vertical, horizontal] que indica el desplazamiento vertical y horizontal del vecino respecto el origen:



Vecindad =
`[1 0 0; 1 0 0; 1 0 1];`
Localización =
`[-1 -1; 0 -1; 1 -1; 1 1];`

Figura f.196 - Elemento estructurante con vecindad y localizaciones

Sintaxis

```
[offsets, heights] = getneighbors(SE)
```

Descripción

[offsets, heights] = getneighbors(SE) devuelve las localizaciones relativas y las alturas correspondientes a cada uno de los vecinos en el elemento estructurante SE. offsets es un array de P-por-N donde P es el número de vecinos en el elemento estructurante y N es la dimensión. Cada fila de offsets contiene la localización del vecino correspondiente, relativo al centro del elemento estructurante. heights es un vector de P columnas que contiene la altura de cada vecino del elemento estructurante. SE es un objeto STREL. Los valores devueltos en las variables offsets y heights son arrays de valores con precisión double.

Al crear un elemento estructurante con la función strel sin deshabilitar la salida de datos por pantalla mediante el operador ; (punto y coma) siempre se muestra tanto la vecindad como la altura del elemento estructurante creado.

Ejemplos de uso

Ejemplo 1: Crear un elemento estructurante y obtener las localizaciones y alturas de sus vecinos.

```
se = strel([1 0 1],[5 0 -5])
[localizaciones,alturas] = getneighbors(se)

se =
Nonflat STREL object containing 2 neighbors.

Neighborhood:
    1      0      1

Height:
    5      0     -5

localizaciones =
    0     -1
    0      1

alturas =
    5     -5
```

Funciones relacionadas

`strel`, `getnhood`, `getheight`

getnhood

Obtención del array de vecindad de un elemento estructurante

Introducción

Los elementos estructurantes son un tipo de objeto definido en MATLAB esencial en las operaciones morfológicas para examinar la imagen de entrada ya que sirven como patrón de búsqueda. Se utiliza la función `strel` para crearlos.

Los elementos estructurantes bidimensionales o planos consisten fundamentalmente en una matriz de 0's y 1's y son normalmente mucho menores que la imagen a procesar. En los elementos estructurantes tridimensionales o no planos los 0's y 1's definen la extensión del elemento estructurante en el plano-x y en el plano-y, y utilizan además valores de altura asociados con cada pixel para definir la tercera dimensión. El pixel central del elemento estructurante, al que se le llama origen, identifica el pixel de interés, el pixel que se está procesando. Los píxeles con valor 1 definen la vecindad del elemento estructurante y se les denomina vecinos del elemento estructurante. La localización de los vecinos se hace referente al pixel central con la notación [vertical, horizontal] que indica el desplazamiento vertical y horizontal del vecino respecto el origen:

1	0	0
1	0	0
1	0	1

Vecindad =
[1 0 0; 1 0 0; 1 0 1];
Localización =
[-1 -1; 0 -1; 1 -1; 1 1];

Figura f.196 - Elemento estructurante con vecindad y localizaciones

Sintaxis

```
NHOOD = getnhood(SE)
```

Descripción

NHOOD = getnhood(SE) devuelve el array de vecindad o patrón asociado al elemento estructurante SE.

SE es un objeto STREL. NHOOD es un array tipo logical.

Ejemplos de uso

Ejemplo 1: Visualizar un elemento estructurante leyendo su array de vecindad.

```
se = strel('disk',4);
nhood = getnhood(se)
imagesc(nhood); axis xy; colormap('gray');
```

```
nhood =
    0      0      1      1      1      0      0
    0      1      1      1      1      1      0
    1      1      1      1      1      1      1
    1      1      1      1      1      1      1
    1      1      1      1      1      1      1
    0      1      1      1      1      1      0
    0      0      1      1      1      0      0
```

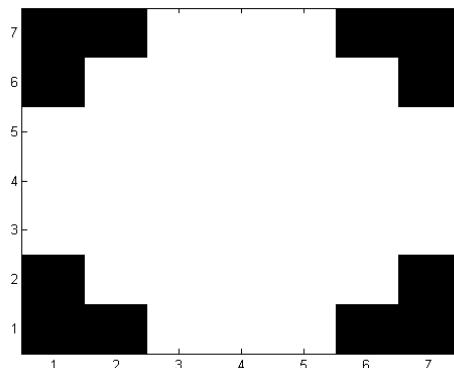


Figura f.197 - Figura elemento estructurante

Funciones relacionadas

strel, getneighbo

getpts

Obtención de coordenadas de puntos en una figura mediante su selección

Sintaxis

```
[x, y] = getpts(fig)  
[x, y] = getpts(ax)  
[x, y] = getpts
```

Descripción

[x, y] = getpts(fig) permite que se especifiquen interactivamente un conjunto de puntos en los ejes actuales de la figura fig utilizando el ratón y recoger sus coordenadas en las variables x e y.

Usar el clic derecho del ratón para añadir puntos. Teniendo presionadas las teclas Shift o derecha al hacer clic, o haciendo doble-clic, se añade un punto final y se finaliza la selección. Presionando Return o Enter se finaliza la selección sin añadir un punto final. Presionando Backspace o Delete se elimina el punto seleccionado.

[x, y] = getpts(ax) permite elegir puntos en los ejes especificados por el puntero ax.

[x, y] = getpts es lo mismo que [x,y] = getpts(gcf).

Funciones relacionadas

getline, getrect

getrangepfromclass

Obtención del rango de visualización por defecto de una imagen según su clase

Sintaxis

```
range = getrangepfromclass(I)
```

Descripción

range = getrangepfromclass(I) devuelve el rango de visualización por defecto de la imagen I, basándose en su tipo o clase, en la variable range, un vector de dos elementos (mínimo, máximo) que especifica el rango de visualización en la forma [min max].

I puede ser de tipo uint8, uint16, int16, logical, single, o double. range es de clase double.

Nota: Para tipos de dato single y double, la función getrangepfromclass devuelve el rango [0,1] para ser consecuente con la forma en la que MATLAB interpreta dichos tipos de imagen. Para datos enteros, getrangepfromclass devuelve el rango de visualización por defecto de la clase. Por ejemplo, si la clase es uint8, el rango dinámico es [0, 255].

Ejemplos de uso

Ejemplo 1: Leer una imagen en formato DICOM (*Digital Imaging and Communication in Medicine*) y obtener su rango de visualización por defecto.

```
CT = dicomread('CT-MONO2-16-ankle.dcm');  
r = getrangepfromclass(CT)  
  
r =  
-32768 32767
```

Funciones relacionadas

intmin, intmax

getrect

Obtención de las coordenadas de un rectángulo en una figura mediante su selección

Sintaxis

```
rect = getrect(fig)
rect = getrect(ax)
```

Descripción

`rect = getrect(fig)` permite que se especifique interactivamente un rectángulo en los ejes actuales de la figura `fig` utilizando el ratón y recoger sus coordenadas en la variable `rect`, un vector de cuatro elementos (mínimo valor en x, mínimo valor en y, anchura y altura) en la forma `[xmin ymin width height]`.

Se debe usar el ratón para hacer clic y seleccionar el rectángulo deseado. Para hacer que el rectángulo sea un cuadrado se debe tener pulsada primero la tecla Shift o hacer clic con el botón derecho para empezar la selección.

`rect = getrect(ax)` permite seleccionar un rectángulo en los ejes actuales especificados por el puntero `ax`.

Funciones relacionadas

`getline`, `getpts`

getsequence

Obtención de los elementos estructurantes necesarios para la descomposición de un elemento estructurante

Introducción

Los elementos estructurantes son un tipo de objeto definido en MATLAB esencial en las operaciones morfológicas para examinar la imagen de entrada ya que sirven como patrón de búsqueda. Se utiliza la función `strel` para crearlos. Más información en las páginas de las funciones `getheight`, `getneighbors` o `getnhood`.

En la dilatación de imágenes, algunos elementos estructurantes pueden ser de mucha utilidad escribiéndolos como la dilatación de dos o más elementos estructurantes más pequeños para ahorrar procesado de datos. Por ejemplo, un array simple `b` de tamaño 5-por-5 puede descomponerse en dos elementos estructurantes más pequeños, una línea horizontal `b1` y una vertical `b2`:

```
b = ones(5, 5)

b =
    1     1     1     1     1
    1     1     1     1     1
    1     1     1     1     1
    1     1     1     1     1
    1     1     1     1     1

b1 = ones(1, 5)

b1 =
    1     1     1     1     1

b2 = ones(5, 1)

b2 =
    1
    1
    1
    1
    1

imdilate(b1, b2, 'full')

ans =
    1     1     1     1     1
    1     1     1     1     1
    1     1     1     1     1
    1     1     1     1     1
    1     1     1     1     1
```

Al realizar la dilatación de una imagen directamente a través de la definición, se necesitarían Q comparaciones por pixel, donde Q es el número de elementos del elemento estructurante. En este caso se necesitarían 25 operaciones por pixel al ser un cuadrado de 5-por-5. Sin embargo, como los elementos estructurantes b_1 y b_2 tienen solo 5 elementos cada uno, se podría dilatar primero por uno y a continuación por el otro, necesitando solamente un 40% de la comparación de píxeles.

Sintaxis

```
SEQ = getsequence(SE)
```

Descripción

`SEQ = getsequence(SE)` devuelve el array de elementos estructurantes `SEQ` con los elementos estructurantes individuales que forman la descomposición de `SE`. `SE` puede ser un array de elementos estructurantes. `SEQ` es equivalente a `SE`, pero los elementos de `SEQ` no tienen descomposición. `SE` y `SEQ` son arrays de objetos estructurantes `STREL`.

Ejemplos de uso

Ejemplo 1: Obtener los elementos estructurantes necesarios para la descomposición de un elemento estructurante cuadrado y confirmarlo mediante la función `imdilate`.

```
% Creamos el elemento estructurante cuadrado.  
b = strel('square',5)  
  
% Vemos que habría una descomposición posible con 2 elementos STREL, así  
% que vamos a ver cuáles son y a mostrar cada uno.  
  
decomp = getsequence(b)  
b1 = getnhood(decomp(1))  
b2 = getnhood(decomp(2))  
  
% Confirmamos que la dilatación con esos dos elementos estructurantes da  
el elemento estructurante original.  
  
imdilate(1, decomp, 'full')  
  
b =  
Flat STREL object containing 25 neighbors.  
Decomposition: 2 STREL objects containing a total of 10 neighbors  
  
Neighborhood:  
1 1 1 1 1  
1 1 1 1 1
```

```
1      1      1      1      1  
1      1      1      1      1  
1      1      1      1      1  
  
decomp =  
2x1 array of STREL objects  
  
b1 =  
Flat STREL object containing 5 neighbors.  
  
Neighborhood:  
1  
1  
1  
1  
1  
  
b2 =  
Flat STREL object containing 5 neighbors.  
  
Neighborhood:  
1      1      1      1      1  
  
ans =  
1      1      1      1      1  
1      1      1      1      1  
1      1      1      1      1  
1      1      1      1      1  
1      1      1      1      1
```

Ejemplo 2: Obtener las formas en las que se puede descomponer un elemento estructurante diamante de 5 por 5.

```
% Creamos un elemento estructurante diamante y leemos la secuencia de  
descomposición mostrando cada elemento.
```

```
b = strel('diamond',5)  
decomp = getsequence(b)  
for k = 1:numel(decomp)  
    getnhood(decomp(k))  
end  
  
b =  
Flat STREL object containing 61 neighbors.  
Decomposition: 4 STREL objects containing a total of 17 neighbors  
  
Neighborhood:  
0      0      0      0      0      1      0      0      0      0      0  
0      0      0      0      1      1      1      0      0      0      0  
0      0      0      1      1      1      1      1      0      0      0  
0      0      1      1      1      1      1      1      1      0      0  
0      1      1      1      1      1      1      1      1      1      0  
1      1      1      1      1      1      1      1      1      1      1  
0      1      1      1      1      1      1      1      1      1      0  
0      0      1      1      1      1      1      1      1      0      0  
0      0      0      1      1      1      1      1      0      0      0  
0      0      0      0      1      1      1      0      0      0      0  
0      0      0      0      0      1      0      0      0      0      0
```

```
decomp =
4x1 array of STREL objects

ans =
    0     1     0
    1     1     1
    0     1     0

ans =
    0     1     0
    1     0     1
    0     1     0

ans =
    0     0     1     0     0
    0     0     0     0     0
    1     0     0     0     1
    0     0     0     0     0
    0     0     1     0     0

ans =
    0     1     0
    1     0     1
    0     1     0
```

Funciones relacionadas

[imdilate](#), [imerode](#), [strel](#)

gray2ind

Conversión de una imagen binaria o en escala de grises a una imagen indexada

Sintaxis

```
[X, map] = gray2ind(I, n)
[X, map] = gray2ind(BW, n)
```

Descripción

[X, map] = gray2ind(I, n) convierte la imagen en escala de grises I a una imagen indexada X. n especifica el tamaño del mapa de color, que es gray(n). n debe ser entero y estar comprendido entre 1 y 65536. Si se omite n, el valor por defecto es 64.

[X, map] = gray2ind(BW, n) convierte la imagen binaria BW en una imagen indexada X. n especifica el tamaño del mapa de color, que es gray(n). Si se omite n, el valor por defecto es 2.

gray2ind escala y después redondea la imagen de intensidad para producir una imagen indexada equivalente.

La imagen de entrada I puede ser de tipo logical, uint8, uint16, int16, single, o double y debe ser real y no dispersa. La imagen I puede tener cualquier dimensión. La clase de la imagen de salida X es uint8 si la longitud del mapa de color es menor o igual que 256; en caso contrario será tipo uint16.

Ejemplos de uso

Ejemplo 1: Convertir una imagen en escala de grises en una imagen indexada y comparar ambas imágenes.

```
I = imread('cameraman.tif');
imshow(I)
[X, map] = gray2ind(I, 16);
figure, imshow(X, map);
```



Figura f.179 - Imagen original (*I*) en escala de grises



Figura f.198 - Imagen convertida a indexada (*X*)

Funciones relacionadas

`grayslice`, `ind2gray`, `mat2gray`

graycomatrix

Creación de una matriz de coocurrencia de niveles de gris (GLCM) a partir de una imagen

Introducción

La textura de una superficie hace referencia a la distribución de valores de intensidad a nivel espacial. Existen diferentes métodos de caracterización de texturas y entre ellos uno de los más utilizados es el de las medidas estadísticas de segundo orden como las matrices de coocurrencia.

La coocurrencia de una imagen en niveles de gris es la probabilidad de que un pixel sea de nivel de gris i y que sus vecinos direccionales tengan nivel de gris j. La matriz de coocurrencia de niveles de gris GLCM (*Grey Level Co-occurrence Matrix*), es un histograma de los niveles de grises de dos dimensiones para un par de píxeles (píxel de referencia y vecino) y es una herramienta potente que permite analizar la textura de la imagen. Esta matriz considera no solamente la distribución de intensidades, sino también las posiciones de los píxeles que tienen iguales o parecidos valores de intensidad. [15] [16]

La siguiente imagen muestra cómo se calculan varios valores en la GLCM de una imagen I de 4-por-5. El elemento (1,1) en la matriz vale 1 porque en la imagen solo ocurre una vez que dos píxeles adyacentes horizontalmente tengan valores 1 y 1 respectivamente. El elemento (1,2) vale 2 porque en la imagen ocurre dos veces que dos píxeles adyacentes tengan los valores 1 y 2. `graycomatrix` es la función que se encarga de este proceso de relleno de todos los valores en la matriz GLCM.

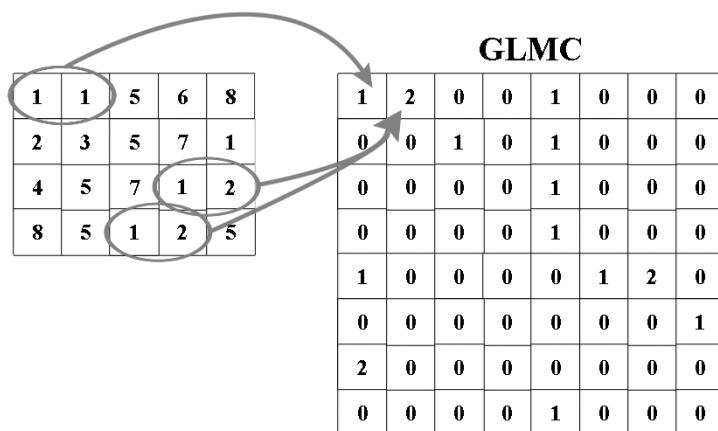


Figura f.199 - Proceso GLMC

Sintaxis

```
glcm = graycomatrix(I)
glcms = graycomatrix(I, param1, val1, param2, val2,...)
[glcm, SI] = graycomatrix(...)
```

Descripción

`glcm = graycomatrix(I)` crea una matriz de co-ocurrencia de niveles de gris (GLCM) a partir de la imagen `I`. `graycomatrix` crea la matriz calculando la frecuencia con la que un pixel con un nivel del gris *i* determinado (intensidad de gris) aparece adyacente de forma horizontal a otro pixel de valor *j*. (Se pueden especificar otras relaciones espaciales utilizando el parámetro `Offsets`). Cada elemento (i,j) en la variable `glcm` especifica el número de veces que el pixel de valor *i* ha aparecido adyacente de forma horizontal al pixel de valor *j*.

`graycomatrix` calcula la matriz GLCM a partir de una versión escalada de la imagen. Por defecto, si `I` es una imagen binaria, `graycomatrix` escala la imagen a dos niveles de gris. Si `I` es una imagen de intensidades, `graycomatrix` escala la imagen a ocho niveles de gris. Se puede especificar el número de niveles de gris que usará la función mediante el parámetro `NumLevels`, y la forma en que la función escala los valores con el parámetro `'GrayLimits'`.

`glcms = graycomatrix(I, param1, val1, param2, val2,...)` devuelve una o más GLCMs dependiendo de los valores de varios parámetros. Los nombres de los parámetros se pueden abbreviar y no son sensibles a las mayúsculas:

Parámetro	Descripción	Por defecto
<code>'GrayLimits'</code>	Vector de dos elementos, <code>[low high]</code> , que especifica cómo se escalan linealmente los niveles de gris. Los valores menores o iguales que <code>low</code> se escalarán a 1, mientras que los valores mayores o iguales a <code>high</code> se escalarán a <code>NumLevels</code> . Si <code>graylimits</code> se pone como <code>[]</code> , <code>graycomatrix</code> usa como límite el mínimo y máximo valor de gris en la imagen: <code>[min(I(:)) max(I(:))]</code> .	Mínimo y máximo especificados por clase p. ej. <code>double [0 1]</code>
<code>'NumLevels'</code>	Entero que especifica el número de niveles de gris que se usará al escalar los valores en escala de gris de la imagen <code>I</code> . Por ejemplo, si <code>NumLevels</code> es 8,	8 (numeric) 2 (binary)

	graycomatrix escala los valores en I de forma que solo hay valores enteros del 1 al 8. El número de niveles de gris determina el tamaño de la matriz de co-ocurrencia (glcm).											
'Offset'	<p>Array de enteros de tamaño $p \times 2$ que especifica la distancia entre el pixel de interés y su vecino. Cada fila en el array es un vector de dos elementos [row_offset, col_offset], que especifica la posición o desfase de un par de píxeles, referida desde el pixel de interés.</p> <p>row_offset es el número de filas de distancia o desfase entre el pixel de interés y su vecino. col_offset es el número de columnas de distancia entre el pixel de interés y su vecino. Los valores positivos son abajo y a la derecha, mientras que los negativos representan el desfase para arriba y para la izquierda.</p> <p>Dado que el desfase se expresa normalmente como un ángulo, la tabla siguiente muestra la equivalencia para valores de ángulos habituales, para una distancia de pixel de D.</p> <table style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th style="text-align: center;">Ángulo</th> <th style="text-align: center;">Desfase</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">[0 D]</td> </tr> <tr> <td style="text-align: center;">45</td> <td style="text-align: center;">[-D D]</td> </tr> <tr> <td style="text-align: center;">90</td> <td style="text-align: center;">[-D 0]</td> </tr> <tr> <td style="text-align: center;">135</td> <td style="text-align: center;">[-D -D]</td> </tr> </tbody> </table> <p>P. ej. Un array [2 0] significa que cada pixel se compara con el pixel dos filas más abajo y en la misma columna. Un array [2 0; 0 2] significa que cada pixel se compara con dos vecinos: el pixel dos filas más abajo y el pixel dos columnas más a la derecha.</p> <p>La figura siguiente representa el array, en el cual cada pixel de interés se compara con 4 vecinos:</p> <pre>offset = [0 1; -1 1; -1 0; -1 -1]</pre>	Ángulo	Desfase	0	[0 D]	45	[-D D]	90	[-D 0]	135	[-D -D]	[0 1]
Ángulo	Desfase											
0	[0 D]											
45	[-D D]											
90	[-D 0]											
135	[-D -D]											

	<p>Figura f.200 - Offset</p>	
'Symmetric'	<p>Booleano que crea una GLCM donde no se considera el orden de los valores en los pares de píxeles. Por ejemplo, cuando se pone 'Symmetric' a true, graycomatrix cuenta los pares 1,2 y 2,1 al calcular el número de veces que el 1 es adyacente al 2. Cuando 'Symmetric' se pone a false, graycomatrix solo cuenta 1,2 o 2,1, dependiendo del valor de 'offset'.</p>	false

[glcm, SI] = graycomatrix(...) devuelve la imagen escalada SI que se usó para calcular la GLCM. Los valores de SI están comprendidos entre 1 y NumLevels.

I puede ser de tipo numeric o logical pero debe ser bidimensional, real, y no dispersa. SI es una matriz en formato double que tiene el mismo tamaño que I. glcms es un array double de tamaño 'NumLevels'-por-'NumLevels'-por-P donde P es el número de desfases en el parámetro Offset.

Notas: Otro nombre para la matriz de coocurrencia de niveles de gris o GLCM es una matriz de dependencia espacial de niveles de gris.

graycomatrix ignora los pares de pixel si cualquiera de los píxeles contiene un número indeterminado NaN (*Not-a-Number*).

graycomatrix remplaza Inf's (infinitos) positivos con el valor NumLevels e Inf's (infinitos) negativos con el valor 1.

graycomatrix ignora píxeles en los bordes si el pixel vecino correspondiente cae fuera de las fronteras de la imagen.

La GLCM creada cuando se usa el parámetro 'Symmetric' puesto a true es simétrica a través de su diagonal. La GLCM producida por la siguiente sintaxis cuando 'Symmetric' vale true:

```
graycomatrix(I, 'offset', [0 1], 'Symmetric', true)
```

es equivalente a la suma de dos GLCMs producidas por las siguientes expresiones cuando el parámetro 'Symmetric' vale false.

```
graycomatrix(I, 'offset', [0 1], 'Symmetric', false)
```

```
graycomatrix(I, 'offset', [0 -1], 'Symmetric', false)
```

Ejemplos de uso

Ejemplo 1: Calcular la matriz de coocurrencia de niveles de gris (GLCM) de una imagen en escala de grises.

```
I = imread('cameraman.tif');  
glcm = graycomatrix(I,'Offset',[2 0]);
```

Funciones relacionadas

[graycoprops](#)

graycoprops

Obtención de propiedades de una matriz de co-ocurrencia de niveles de gris (GLCM)

Introducción

La coocurrencia de una imagen en niveles de gris es la probabilidad de que un pixel sea de nivel de gris i y que sus vecinos direccionales tengan nivel de gris j . La matriz de coocurrencia de niveles de gris GLCM (*Grey Level Co-occurrence Matrix*), es un histograma de los niveles de grises de dos dimensiones para un par de píxeles (píxel de referencia y vecino) y es una herramienta potente que permite analizar la textura de la imagen. Esta matriz considera no solamente la distribución de intensidades, sino también las posiciones de los píxeles que tienen iguales o parecidos valores de intensidad. Más información en la página dedicada a la función `graycomatrix`.^[15]^[16]

Sintaxis

```
stats = graycoprops(glcm, properties)
```

Descripción

`stats = graycoprops(glcm, properties)` calcula estadísticas especificadas en la variable `properties` a partir de una matriz de coocurrencia de niveles de gris `glcm` que es un array de m -por- n -por- p de matrices de co-ocurrencia válidas. Si `glcm` es un array de matrices de coocurrencia, `stats` es un array de estadísticas para cada `glcm`.

`graycoprops` normaliza la GLCM para que la suma de sus elementos sea igual a 1. Cada elemento (r,c) en la matriz normalizada es la probabilidad de que el pixel de nivel de gris i aparezca con una relación espacial definida (p. ej. adyacente de forma horizontal) al pixel de nivel j . `graycoprops` usa la matriz normalizada para calcular las estadísticas definidas en `properties`.

`properties` puede ser una lista de propiedades separadas por coma, un array de celdas con propiedades, la propiedad `'all'` o propiedades separadas por espacios en blanco. Los nombres de las propiedades, listados a continuación, se pueden abreviar y no son sensibles a las mayúsculas:

Propiedad	Descripción
'Contrast'	Devuelve una medida de contraste de intensidad entre un pixel y su vecino, sobre la imagen completa. Rango = [0 (size(GLCM, 1)-1)^2] Su valor será 0 para una imagen constante. También conocida como varianza e inercia.
'Correlation'	Devuelve una medida de cómo está un pixel de correlado a su vecino, sobre la imagen completa. Rango = [-1,1] Su valor será 1 o -1 para una imagen perfectamente correlada positiva o negativamente. Será un número indeterminado NaN (<i>Not-a-Number</i>) para una imagen constante.
'Energy'	Devuelve la suma de los elementos encuadrados en la GLCM. Rango = [0,1] Su valor será 1 para una imagen constante. También conocida como uniformidad y segundo momento angular.
'Homogeneity'	Devuelve un valor que mide el parecido de la distribución de elementos en la GLCM a la diagonal de la matriz. Rango = [0,1] Su valor será 1 para una GLCM diagonal.

`stats` es una estructura con campos especificados por `properties`. Cada campo contiene un array de $1 \times p$, donde p es el número de GLCMs. Por ejemplo, si `GLCM` es un array de $8 \times 8 \times 3$ y la propiedad es '`Energy`', la variable `stats` será una estructura que contenga el campo '`Energy`' que a su vez contendrá un array de 1×3 .

`glcm` puede ser de tipo `numeric` o `logical` y debe contener enteros reales, no negativos y finitos. `stats` es una estructura.

Ejemplos de uso

Ejemplo 1: Crear una matriz de co-ocurrencia y obtener sus propiedades.

```
GLCM = [0 1 2 3;1 1 2 3;1 0 2 0;0 0 0 3];
stats = graycoprops(GLCM)
```

```
stats =
    Contrast: 2.8947
    Correlation: 0.0783
    Energy: 0.1191
```

Homogeneity: 0.5658

Ejemplo 2: Crear una matriz de co-ocurrencia a partir de una imagen y obtener sus propiedades.

```
I = imread('circuit.tif');
GLCM2 = graycomatrix(I,'Offset',[2 0;0 2]);
stats = graycoprops(GLCM2,['contrast','homogeneity'])

stats =
    Contrast: [0.3307 0.3358]
    Homogeneity: [0.8534 0.8531]
```

Funciones relacionadas

`graycomatrix`

grayslice

Conversión de una imagen en escala de grises a una imagen indexada utilizando umbral multinivel

Sintaxis

```
X = grayslice(I, n)
```

Descripción

`X = grayslice(I, n)` devuelve una imagen indexada `X` a partir de una imagen de intensidades `I` utilizando los valores de umbral de valores: $\frac{1}{n}, \frac{2}{n}, \dots, \frac{n-1}{n}$

`X = grayslice(I, v)` umbraliza la imagen indexada `I` utilizando los valores de `v` que es un vector de valores entre 0 y 1, devolviendo una imagen indexada en `X`.

La imagen de entrada `I` puede ser de tipo `uint8`, `uint16`, `int16`, `single`, o `double` y debe ser no dispersa. Los valores del umbral están siempre entre 0 y 1, incluso si `I` es del tipo `uint8` o `uint16`. En este caso, cada valor de umbral se multiplica por 255 o 65535 para determinar el umbral real a usar.

La clase de la imagen de salida `X` depende del número de valores de umbral. Si el número de valores de umbral es menor que 256, entonces `X` será de clase `uint8` y sus valores irán desde 0 a `n` o `length(v)`. Si el número de valores de umbral es 256 o más, `X` será de clase `double` y los valores en `X` irán desde 1 hasta `n+1` o `length(v)+1`.

Se puede visualizar la imagen umbralizada utilizando la función `imshow(X, map)` con un mapa de color `map` de longitud apropiada.

Ejemplos de uso

Ejemplo 1: Convertir una imagen en escala de grises a una imagen indexada utilizando umbral multinivel.

```
I = imread('cameraman.tif');
X = grayslice(I,16);
imshow(I)
figure, imshow(X,jet(16))
```



Figura f.179 - Imagen original (I) en escala de grises

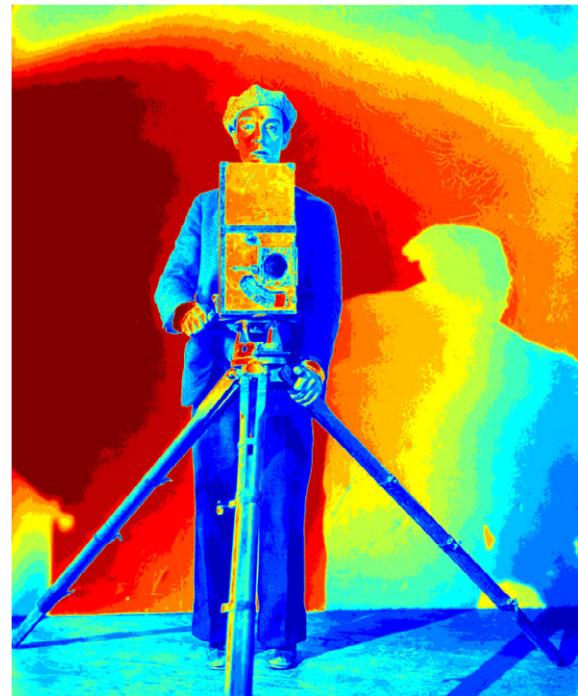


Figura f.201 - Imagen convertida a indexada (X) usando grayslice

Funciones relacionadas

`gray2ind`

graythresh

Cálculo del umbral óptimo de una imagen mediante el método de Otsu

Introducción

La segmentación es un proceso de agrupación de los píxeles de una imagen en regiones determinadas, normalmente conectadas. Es un paso importante (y normalmente necesario) en muchas ramas del procesado de imágenes. Las técnicas de segmentación se pueden dividir en tres grandes grupos: métodos basados en intensidad (p. ej. la umbralización), métodos basados en regiones (p. ej. crecimiento de regiones, división de regiones y fundido de regiones) y otros métodos (segmentación basada en textura, bordes y movimiento).

La umbralización es una técnica para el procesado de imágenes en la que una imagen de entrada en escala de grises se recuantifica a dos niveles de gris, o lo que es lo mismo, se convierte a imagen binaria. Cada pixel de la imagen original se compara con un umbral y el resultado de esa comparación determina si el pixel se convertirá a un nivel o a otro (blanco o negro, objeto o fondo). El algoritmo de umbralización más simple es aquél que usa un solo valor de umbral para toda la imagen (umbral global, `im2bw` en MATLAB). ^[17]

La umbralización se suele emplear cuando hay una clara diferencia entre los objetos a extraer respecto del fondo de la escena. Al aplicar un umbral T , la imagen en escala de grises $f(x,y)$ quedará binarizada y etiquetada con ‘1’ los píxeles correspondientes al objeto y con ‘0’ aquellos que son del fondo. Si el umbral solo depende de $f(x,y)$ se dice que es un umbral global; en el caso de que además dependa de $p(x,y)$, por ejemplo, el valor medio de los píxeles vecinos, el umbral se denomina local; y si depende también de la posición (x,y) del pixel, se denomina dinámico.

El problema de la umbralización reside en la determinación del valor umbral que segmenta la imagen. El método de Otsu es uno de los más utilizados en la determinación automática de dicho umbral. Este método proporciona el umbral óptimo para la segmentación de la imagen bajo el criterio de máxima varianza entre fondo y objeto. Así, se calcula la varianza entre todas las posibles divisiones y se toma el umbral que presenta la máxima varianza entre clases. Este método es uno de los mejores métodos de selección de umbral para imágenes del mundo real y se ha establecido como estándar. Además, el método de Otsu es

automático, es decir, no necesita supervisión humana ni información previa de la imagen antes de su procesado.^[18]

Sintaxis

```
level = graythresh(I)
[level EM] = graythresh(I)
```

Descripción

`level = graythresh(I)` calcula un umbral óptimo y lo guarda en la variable `level`. El umbral se puede utilizar para convertir una imagen de intensidades a una imagen binaria con la función `im2bw`. `level` es un valor de intensidad normalizado que está en el rango `[0, 1]`.

La función `graythresh` usa el método de Otsu, el cual elige el umbral óptimo para minimizar la varianza intraclasa de los píxeles blanco y negro.

Los arrays multidimensionales se convierten automáticamente a arrays en 2-D utilizando la función `reshape`. La función `graythresh` ignora cualquier imaginario distinto de cero en la imagen `I`.

`[level EM] = graythresh(I)` devuelve la medida de efectividad `EM` como segundo argumento de salida. La medida de efectividad es un valor en el rango `[0, 1]` que indica la eficacia al umbralizar la imagen de entrada. El límite inferior solo lo alcanzan imágenes con nivel único de gris y el límite superior solo lo alcanzan imágenes con dos niveles.

La imagen de entrada `I` puede ser de tipo `uint8`, `uint16`, `int16`, `single`, o `double` y debe ser de tipo no disperso. El valor devuelto `level` es un escalar de tipo `double`, igual que la medida de efectividad `EM`.

Ejemplos de uso

Ejemplo 1: Convertir una imagen en escala de grises a binario utilizando la función `im2bw` y el umbral obtenido por el método de Otsu.

% Leemos una imagen en escala de grises y la convertimos a binaria.

```
I = imread('euros.tif');  
imshow(I)  
level = graythresh(I)  
BW = im2bw(I,level);  
figure, imshow(BW)  
  
level =  
0.4078
```



Figura f.202- Imagen original (I) en escala de grises



Figura f.203 - Imagen convertida a binaria (X)

Funciones relacionadas

im2bw

hdrread

Lectura de una imagen de alto rango dinámico (HDR)

Introducción

En procesado de imágenes, gráficos por ordenador y fotografía, las imágenes de alto rango dinámico (HDR, *High Dynamic Range*) son un conjunto de técnicas que permiten un mejor rango dinámico de luminancias entre las zonas más claras y las más oscuras de una imagen del que pueden ofrecer las técnicas de imagen digital estándar o métodos fotográficos. Este rango dinámico más extenso permite a las imágenes HDR representar con más exactitud el extenso rango de niveles de intensidad encontrados en escenas reales, que van desde luz solar directa hasta la débil luz de las estrellas.^[19]

Los dos principales orígenes de las imágenes HDR son el renderizado por ordenador y la mezcla de múltiples fotografías, que a su vez son conocidas como fotografías de bajo rango dinámico (LDR), también llamadas de rango dinámico estándar (SDR).^[20]

Las técnicas de mapeo de tonos, que reducen todo el contraste para facilitar que dispositivos con menos rango dinámico muestren imágenes HDR, pueden aplicarse para producir imágenes conservando o exagerando el contraste localmente para realizar un efecto artístico.

Sintaxis

```
hdr = hdrread(filename)
```

Descripción

`hdr = hdrread(filename)` lee una imagen de alto rango dinámico especificada en la variable `filename`. `hdr` es un array RGB de m-por-n-por-3 en el rango `[0, inf)` y del tipo `single`. Para datos referidos a una escena, estos valores suelen ser la iluminación de la escena en unidades de radiancia espectral. Para mostrar estas imágenes se debe usar un operador de mapeo tonal apropiado.

La imagen de salida `hdr` es una imagen de m-por-n-por-3 del tipo `single`.

Ejemplos de uso

Ejemplo 1: Leer y mostrar una imagen de alto rango dinámico.

```
hdr = hdrread('office.hdr');  
I = tonemap(hdr);  
imshow(I)
```

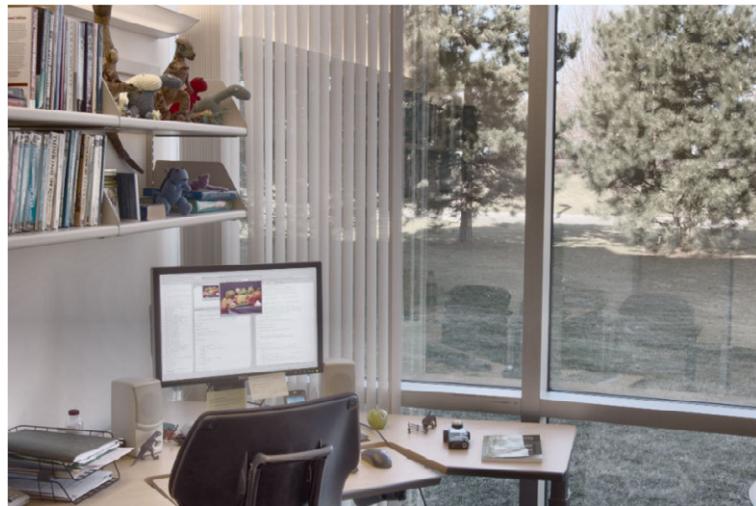


Figura f.204 - Imagen hdr en RGB (I)

Funciones relacionadas

`hdrwrite`, `makehdr`, `tonemap`

hdrwrite

Escribir una imagen HDR en un archivo en formato .hdr (*Radiance*)

Introducción

Radiance (.hdr) es un formato de archivo de 32 bits por canal utilizado en imágenes de alto rango dinámico (HDR, *High Dynamic Range*). Este formato se desarrolló originalmente para el sistema Radiance, una herramienta profesional para visualizar la iluminación en entornos virtuales. El formato de archivo guarda la cantidad de luz por píxel en lugar de solo los colores que se van a visualizar en la pantalla. Los niveles de luminosidad que es capaz de interpretar el formato Radiance son mucho más altos que los 256 niveles de los formatos de archivos de imágenes de 8 bits por canal. La gama tonal registrada es tan extensa que no es posible representar la imagen en un monitor común. Existen programas específicos que permiten manejar estos archivos y comprimir la gama tonal para poder representar toda esa información, como es el caso de MATLAB. Los archivos Radiance (.hdr) se utilizan normalmente en modelado en 3D.

Sintaxis

```
hdrwrite(hdr, filename)
```

Descripción

`hdrwrite(hdr, filename)` crea un archivo de imagen en formato Radiance .hdr a partir de una imagen HDR de precisión simple o doble presente en el espacio de trabajo de MATLAB. El archivo HDR generado con el nombre `filename` usa compresión RLE (*run-length encoding*) para minimizar el tamaño del archivo.

Funciones relacionadas

`hdrread`, `makehdr`, `tonemap`

histeq

Mejora del contraste mediante ecualización del histograma

Introducción

La función `histeq` mejora el contraste de la imagen `I` utilizando una ecualización del histograma. Es útil para mejorar el contraste global de una imagen aunque puede cambiar el brillo de la imagen final o sobreexponer algunas zonas de la imagen. Otras alternativas son las funciones `adapthisteq`, que ecualiza el histograma de forma adaptativa actuando sobre pequeñas zonas e histogramas y obteniendo mejores resultados en general, e `imadjust`, que permite expandir el histograma ajustando los valores de intensidad de la imagen a un rango determinado. Más información y comparación en la página de la función `adapthisteq`.

Sintaxis

```
J = histeq(I, hgram)
J = histeq(I, n)
[J, T] = histeq(I,...)
newmap = histeq(X, map, hgram)
newmap = histeq(X, map)
[newmap, T] = histeq(X,...)
```

Descripción

`histeq` mejora el contraste de las imágenes transformando los valores de una imagen de intensidades o los valores del mapa de color de una imagen indexada para que el histograma de la imagen de salida se aproxime al histograma especificado.

`J = histeq(I, hgram)` transforma la imagen de intensidades `I` de forma que el histograma de la imagen de salida `J` de longitud `length(hgram)` concuerde de forma aproximada con el histograma especificado en la variable `hgram`. El vector `hgram` debe contener enteros equiespaciados con valores de intensidad en el rango apropiado: [0, 1] para imágenes de tipo `double`, [0, 255] para imágenes de tipo `uint8` y [0, 65535] para imágenes de tipo `uint16`. `histeq` escala automáticamente `hgram` para que `sum(hgram) = prod(size(I))`. El histograma de `J` se ajustará más a `hgram` cuando `length(hgram)` sea mucho menor que el número de niveles discretos en `I`.

`J = histeq(I, n)` transforma la imagen de intensidades `I` devolviendo una imagen de intensidades `J` con `n` niveles de gris discretos. Se mapea en `J` un número de píxeles aproximadamente igual en cada uno de los `n` niveles de forma que el histograma de `J` sea aproximadamente plano (el histograma de `J` será más plano cuando `n` sea mucho más pequeño que el número de niveles discretos en `I`). El valor por defecto de `n` es 64.

`[J, T] = histeq(I, ...)` devuelve la transformación de escala de grises `T` que mapea los niveles de gris de la imagen `I` en niveles de gris en la imagen `J`.

`newmap = histeq(X, map, hgram)` transforma el mapa de color `map` asociado con la imagen indexada `X` tal que el histograma de la componente gris en la imagen indexada `(X, newmap)` se ajusta aproximadamente al histograma de la variable `hgram`. La función `histeq` devuelve el mapa de color transformado en la variable `newmap`. `length(hgram)` debe ser igual que `size(map, 1)`.

`newmap = histeq(X, map)` transforma los valores del mapa de color `map` para que el histograma de la componente gris de la imagen indexada `X` sea aproximadamente plano. Devuelve el mapa de color transformado en la variable `newmap`.

`[newmap, T] = histeq(X, ...)` devuelve la transformación en escala de grises `T` que mapea la componente gris de `map` en la componente gris de `newmap`.

Para las sintaxis que incluyen como entrada una imagen de intensidades `I`, la imagen puede ser del tipo `uint8`, `uint16`, `int16`, `single`, o `double`. La imagen de salida `J` tendrá el mismo tipo que `I`. Para la sintaxis que incluye como entrada una imagen indexada `X`, la imagen puede ser del tipo `uint8`, `single`, o `double`. El mapa de color de salida es siempre de tipo `double`. La salida opcional `T` (la transformación de los niveles de gris) es siempre de tipo `double`.

Ejemplos de uso

Ejemplo 1: Mejorar el contraste de una imagen con una ecualización del histograma y comparar los histogramas original y mejorado.

```
I = imread('M131806467LC.tif');
J = histeq(I);
imshow(I); figure, imshow(J)
```

```
figure, imhist(I); figure, imhist(J)
```



Figura f.1 - Imagen original (I)

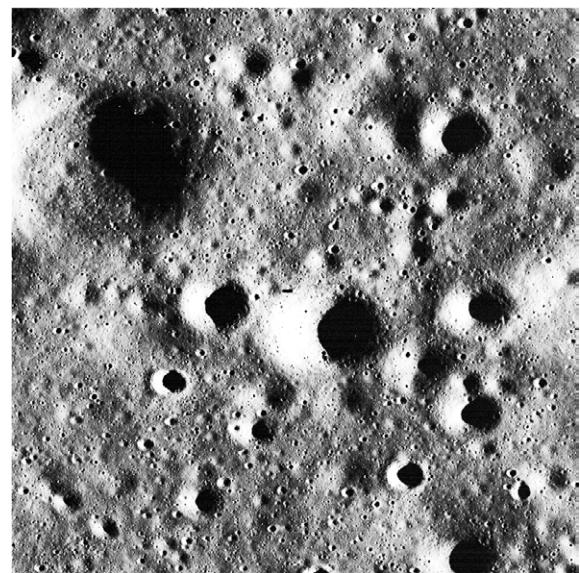


Figura f.6 - Imagen con contraste mejorado (J)

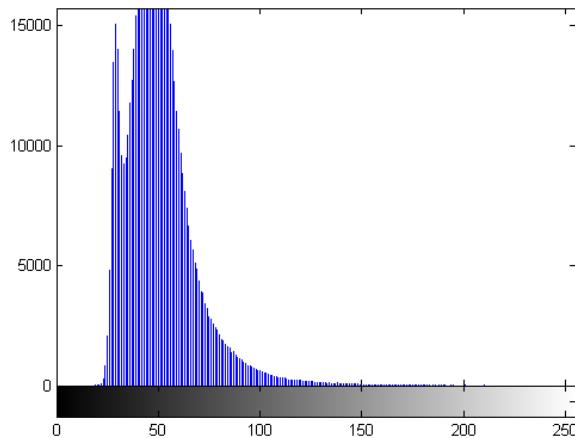


Figura f.7 - Histograma de imagen original (I)

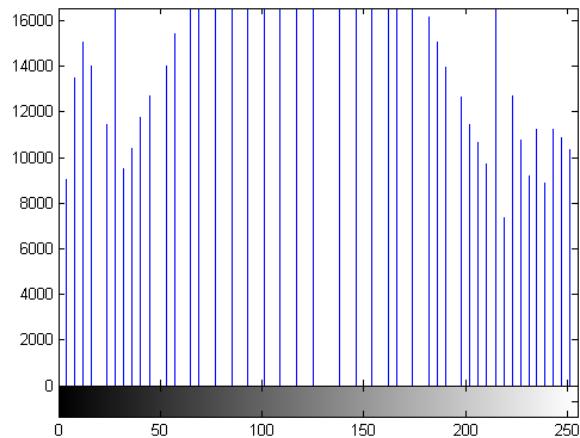


Figura f.9 - Histograma de imagen con contraste mejorado (J)

Funciones relacionadas

adaphisteq, brighten, imadjust, imhist

hough

Transformada Hough

Introducción

La transformada de Hough permite, tras procesar una imagen, detectar todas las líneas rectas dentro de ella (u otros objetos geométricos que admitan descripciones paramétricas). Es una técnica muy robusta frente al ruido y a la existencia de huecos en la frontera del objeto. La idea básica de esta técnica es encontrar curvas tales como rectas, polinomios, círculos, etc., que puedan ser localizadas en un espacio de parámetros adecuado. Aunque la transformada se puede utilizar para dimensiones mayores, se usa principalmente en dos dimensiones y es muy recurrida en técnicas de segmentación.

La función `hough` en MATLAB se encarga de realizar la Transformada de Hough. La función devuelve una matriz `H` en la que el valor de cada una de sus posiciones se corresponde con el número de píxeles que comparten un determinado valor de `rho`, alineados en la dirección perpendicular a `theta`, es decir, el ángulo que forman las rectas y su distancia perpendicular hasta el origen de coordenadas. La línea definida por `theta = 60°` y `rho = 2` atravesará un número de píxeles determinado. Cuanto mayor sea el número de píxeles, mayor será la probabilidad de que haya una línea principal. En la representación gráfica de la matriz se observará una elevada intensidad en esas posiciones debido a la concentración de píxeles en esas coordenadas. En general, los valores máximos de la matriz `H` se corresponden con las líneas detectadas.

MATLAB dispone de dos funciones para el cálculo de picos y de líneas a partir de la matriz de Hough `H`. Estas son `houghpeaks`, para hallar las coordenadas de los máximos de la matriz `H` y `houghlines`, que dibuja los segmentos mayores correspondientes a los máximos devueltos por la función `houghpeaks`.

Sintaxis

```
[H, theta, rho] = hough(BW)
[H, theta, rho] = hough(BW, ParámetroName, ParámetroValor)
```

Descripción

[*H*, *theta*, *rho*] = *hough*(*BW*) calcula la Transformada de Hough estándar (SHT) de la imagen binaria *BW*. Usa la función *hough* para detectar líneas en la imagen. La función devuelve *H*, la matriz de la Transformada de Hough. *theta* (en grados) y *rho* son los arrays de las variables *rho* y *theta* sobre los que la función *hough* genera la matriz. *BW* puede ser de tipo *numeric* o *logical* y debe ser real, bidimensional y no disperso.

[*H*, *theta*, *rho*] = *hough*(*BW*, *ParámetroName*, *ParámetroValor*) calcula la Transformada de Hough estándar utilizando pares de parámetros/valores. Cuando uno de los parámetros especificados en la variable *ParámetroName* es 'RhoResolution', se debe definir un valor escalar real entre 0 y *norm(size(BW))* para determinar el espaciado o resolución de la transformada en el eje *rho* y cuyo valor por defecto es 1. Cuando el parámetro especificado en la variable *ParámetroName* es 'ThetaResolution', se debe definir un vector para los valores de *theta* a usar en la Transformada de Hough. El rango aceptado por los valores de *theta* va de $-90^\circ \leq \theta < 90^\circ$, y su valor por defecto es de -90:89.

Ejemplos de uso

Ejemplo 1: Teóricamente por un punto pueden pasar infinitas líneas. Aplicar la transformada de Hough a una imagen con un solo punto sobre fondo negro y mostrar la transformada en el espacio [*theta*,*rho*]. Repetir la operación sobre una imagen con tres puntos separados y alineados y observar el punto donde puede haber una línea a partir de la representación de la Transformada de Hough.

```
I1 = imread('1punto.png');
BW1=im2bw(I1);
imshow(BW1)

% Aplicamos la Transformada de Hough y mostramos.

[H1, thetal, rho1] = hough(BW1);
fh = figure;
imshow(imadjust(mat2gray(H1)), 'XData', thetal, 'YData', rho1,
'InitialMag', 'fit')
xlabel('\theta'), ylabel('\rho');
axis on, axis normal, hold on;
waitfor(fh);

% Repetimos para la otra imagen.
```

```
hold
close all
I3 = imread('3punto.png');
BW3=im2bw(I3);
imshow(BW3)

[H3, theta3, rho3] = hough(BW3);
figure, imshow(imadjust(mat2gray(H3)), 'XData', theta3, 'YData', rho3,
'InitialMag', 'fit')
xlabel('\theta'), ylabel('\rho');
axis on, axis normal, hold on;

% Existe un punto donde se interceptan las tres curvas sinusoidales, lo
% que indica la posibilidad de que existiera una línea. Para determinar los
% valores de rho y theta que definen esa posible línea habría que usar la
% función houghlines o buscar el valor de píxeles más elevado en la matriz
% de Transformada Hough.
```

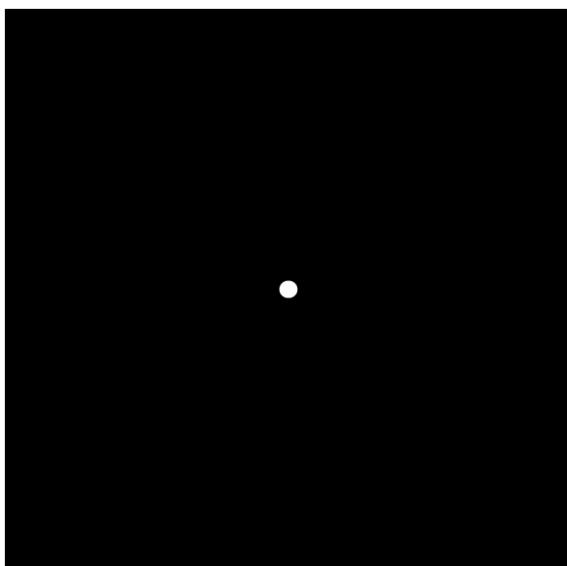


Figura f.205 - Imagen original (I2)

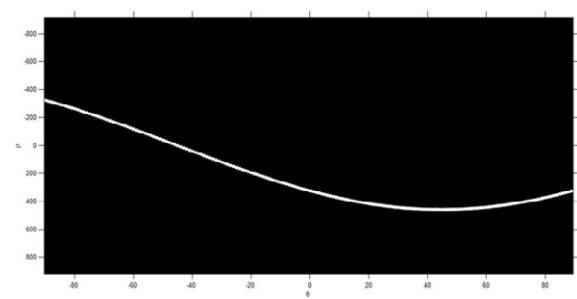


Figura f.206 - Transformada Hough de la imagen I2

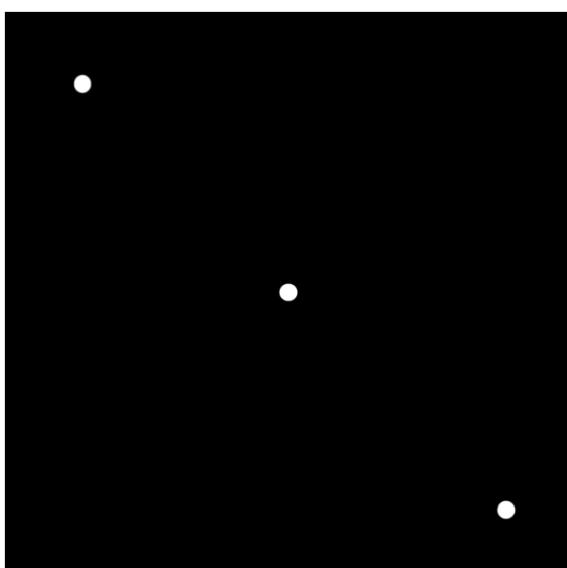


Figura f.207 - Imagen original (I3)

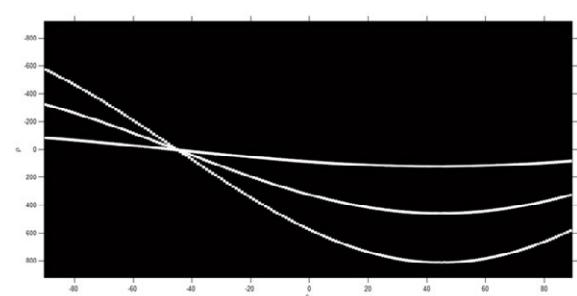


Figura f.208 - Transformada Hough de la imagen I3

Ejemplo 2: Calcular y mostrar la Transformada Hough de una imagen variando la resolución de theta y rho usando los valores 10, 20, 30 y 60.

```
I = imread('3punto.png');
BW=im2bw(I);

[H,T,R] = hough(BW,'RhoResolution',10,'ThetaResolution',10);
figure,
imshow(imadjust(mat2gray(H)),'XData',T,'YData',R,'InitialMagnification','fit')
xlabel('\theta'), ylabel('\rho'); axis on, axis normal;

[H,T,R] = hough(BW,'RhoResolution',20,'ThetaResolution',20);
figure,
imshow(imadjust(mat2gray(H)),'XData',T,'YData',R,'InitialMagnification','fit')
xlabel('\theta'), ylabel('\rho'); axis on, axis normal;

[H,T,R] = hough(BW,'RhoResolution',30,'ThetaResolution',30);
figure,
imshow(imadjust(mat2gray(H)),'XData',T,'YData',R,'InitialMagnification','fit')
xlabel('\theta'), ylabel('\rho'); axis on, axis normal;

[H,T,R] = hough(BW,'RhoResolution',40,'ThetaResolution',60);
figure,
imshow(imadjust(mat2gray(H)),'XData',T,'YData',R,'InitialMagnification','fit')
xlabel('\theta'), ylabel('\rho'); axis on, axis normal;
```

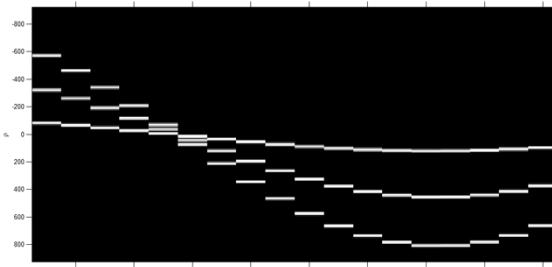


Figura f.209 - Transformada Hough resolución 10

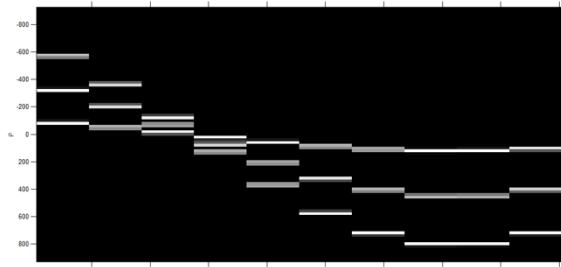


Figura f.210 - Transformada Hough resolución 20

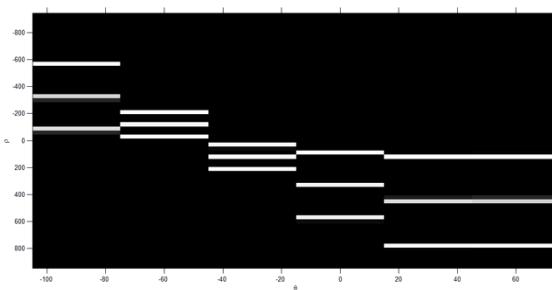


Figura f.211 - Transformada Hough resolución 30

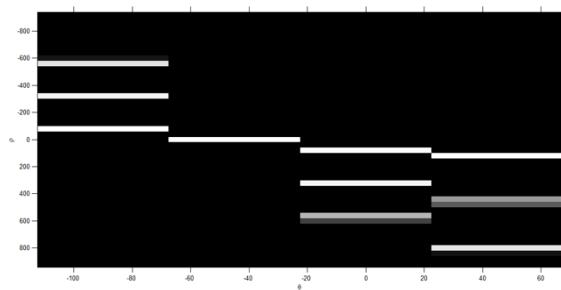


Figura f.212 - Transformada Hough resolución 60

Funciones relacionadas

houghlines, houghpeaks

houghlines

Extracción de segmentos de línea partir de la transformada de Hough

Sintaxis

```
lines = houghlines(BW, theta, rho, peaks)
lines = houghlines(..., param1, val1, param2, val2)
```

Descripción

`lines = houghlines(BW, theta, rho, peaks)` extrae los segmentos de línea de la imagen `BW` asociados a los picos o máximos de la Transformada Hough. `theta` y `rho` son los vectores devueltos por la función `hough` que aplica la Transformada Hough. `peaks` es una matriz devuelta por la función `houghpeaks` que contiene las coordenadas (fila y columna) de los picos de la Transformada Hough que se deben usar para buscar segmentos de línea. Si en la imagen de entrada no están bien definidos los bordes del fondo, el resultado de la búsqueda de líneas no será el esperado, por lo que se deberá procesar antes la imagen en busca de los bordes con la función `edge`.

La función `houghlines` devuelve el array `lines`, un array de estructuras cuya longitud es igual al número de segmentos de línea encontrados. Cada elemento del array tiene los siguientes campos:

Campo	Descripción
<code>point1</code>	Vector de dos elementos [x y] que especifica las coordenadas del punto final del segmento de línea.
<code>point2</code>	Vector de dos elementos [x y] que especifica las coordenadas del punto final del segmento de línea.
<code>theta</code>	Ángulo en grados asociado en la Transformada Hough.
<code>rho</code>	Posición asociada en la Transformada Hough en el eje <code>rho</code> .

`lines = houghlines(..., param1, val1, param2, val2)` especifica pares de parámetros y valores especificados en la siguiente tabla. Los nombres de los parámetros se pueden abbreviar y no son sensibles a las mayúsculas:

Parámetro	Descripción
'FillGap'	Escalar real positivo que especifica la distancia entre dos segmentos de líneas asociados con la misma celda de Transformada Hough. Cuando la distancia entre los segmentos de línea es menor que el valor especificado, la función <code>houghlines</code> combina los segmentos de línea en un único segmento. (Por defecto: 20)
'MinLength'	Escalar real positivo que especifica si las líneas combinadas deben mantenerse o descartarse. Se descartan las líneas más cortas que el valor especificado. (Por defecto: 40)

El resultado de la función varía enormemente con el valor de los dos parámetros opcionales 'FillGap' y 'MinLength', que deberán ser ajustados para cada problema en particular.

`BW` puede ser de tipo `numeric` o `logical` y debe ser real, bidimensional y no disperso.

Ejemplos de uso

Ejemplo 1: Detectar los segmentos de línea en la imagen dada.

```

RGB = imread('lineas.png');
imshow(RGB);
BW = im2bw(RGB);
[H,T,R] = hough(BW);

% En la función houghpeaks se suele especificar un número de máximos/picos mayor al número de líneas de la imagen.

P = houghpeaks(H,20);
lineas = houghlines(BW,T,R,P);

% Visualizar las líneas encontradas.

fh = figure;
imshow(BW), hold on
max_len = 0;
for k = 1:length(lineas)
    xy = [lineas(k).point1; lineas(k).point2];
    plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','red');
    plot(xy(1,1),xy(1,2),'x','LineWidth',2,'Color','yellow');
    plot(xy(2,1),xy(2,2),'x','LineWidth',2,'Color','green');
end

% Vemos que hay dos segmentos que no los detecta. Habría que o bien variar el umbral de detección de la función houghpeaks o buscar un theta y rho determinados tales que el vector de picos/ máximos devuelto por houghpeaks tuviera al menos 5 elementos (las 5 posibles líneas). Más detalle en la función houghpeaks.

```

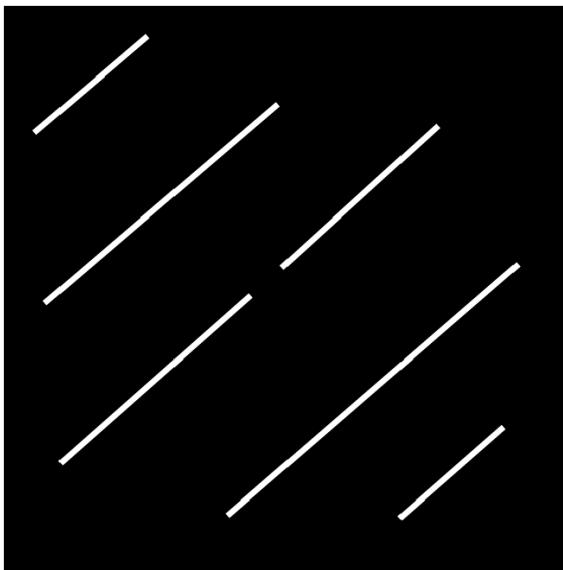


Figura f.213 - Imagen original (RGB)

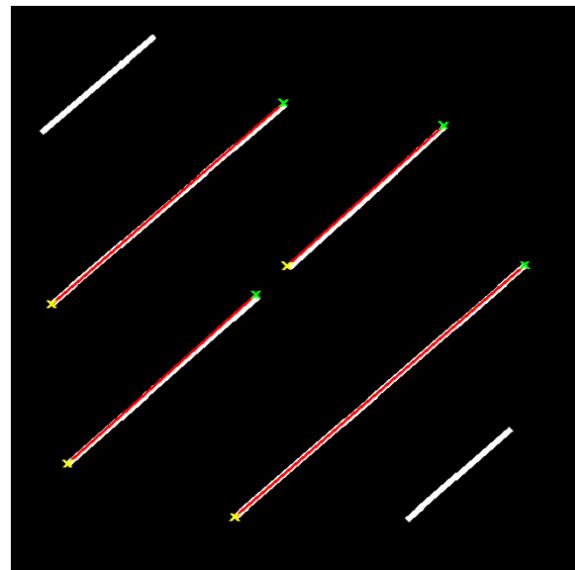


Figura f.214 - Líneas detectadas con la Transformada Hough

Ejemplo 2: Mostrar los segmentos de línea principales de la imagen dada, preparando antes la imagen para mejorar la detección y comparando varios resultados según los parámetros `FillGap` y `MinLength`.

```

RGB = imread('lego.tif');
imshow(RGB)
I = rgb2gray(RGB);

% Pre-procesado para potenciar bordes importantes.

BW = edge(I, 'canny');
BW = bwareaopen(BW, 40);

% Transformada de Hough y búsqueda de máximos.

[H,T,R] = hough(BW);
peaks = houghpeaks(H,100);

% Buscamos las líneas y las mostramos con los valores por defecto.

lines = houghlines(BW,T,R,peaks);
figure, imshow(I), hold on
max_len = 0;
for k = 1:length(lines)
    xy = [lines(k).point1; lines(k).point2];
    plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','red');

    % Dibujamos los principios y finales de las líneas.

    plot(xy(1,1),xy(1,2),'x','LineWidth',2,'Color','yellow');
    plot(xy(2,1),xy(2,2),'x','LineWidth',2,'Color','green');

end

% Cambiamos los valores de FillGap y MinLength para mostrar líneas más
largas.

```

```
lines = houghlines(BW,T,R,peaks,'FillGap',100,'MinLength',10);
figure, imshow(I), hold on
max_len = 0;
for k = 1:length(lines)
    xy = [lines(k).point1; lines(k).point2];
    plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','red');

    % Dibujamos los principios y finales de las líneas.
    plot(xy(1,1),xy(1,2),'x','LineWidth',2,'Color','yellow');
    plot(xy(2,1),xy(2,2),'x','LineWidth',2,'Color','green');

end
```

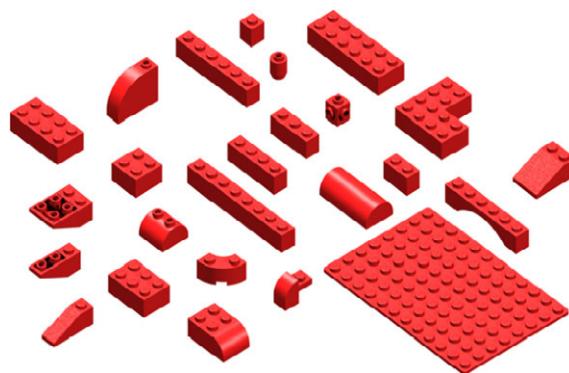


Figura f.215 - Imagen original (RGB)

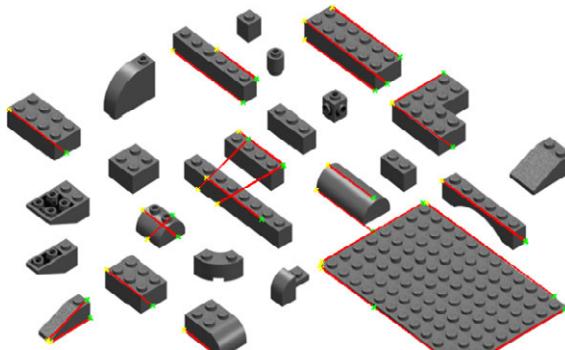


Figura f.216 - Líneas detectadas con la Transformada Hough por defecto

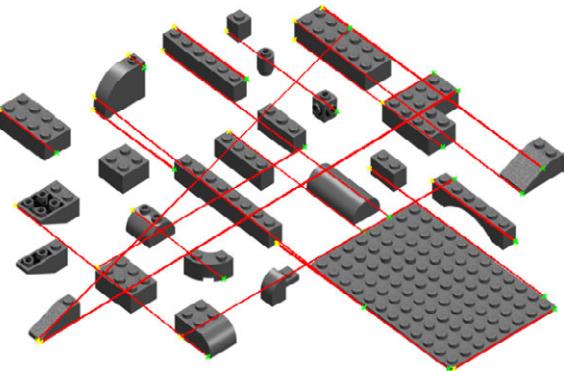


Figura f.217 - Líneas detectadas con la Transformada Hough con 'FillGap'=100 y 'MinLength'=10

Funciones relacionadas

hough, houghpeaks

houghpeaks

Localización de picos en la Transformada Hough

Sintaxis

```
peaks = houghpeaks(H, numpeaks)
peaks = houghpeaks(..., param1, val1, param2, val2)
```

Descripción

`peaks = houghpeaks(H, numpeaks)` localiza los picos o máximos en la matriz de la Transformada Hough `H`, generada por la función `hough`. `numpeaks` es un valor escalar que especifica el máximo número de picos a identificar. Si se omite `numpeaks`, por defecto vale 1.

La función devuelve en `peaks` una matriz Q-por-2 donde Q puede estar comprendido en el rango de 0 a `numpeaks`. Q guarda las coordenadas de la fila y la columna de los picos.

`peaks = houghpeaks(..., param1, val1, param2, val2)` especifica determinados pares de parámetros y valores de entre los listados en la siguiente tabla. Los nombres de los parámetros se pueden abbreviar y no son sensibles a las mayúsculas:

Parámetro	Descripción
'Threshold'	Valor escalar no negativo que especifica el umbral en el que los valores de <code>H</code> se consideran picos. El umbral puede variar desde 0 hasta <code>Inf</code> (infinito). (Por defecto: <code>0.5 * max(H(:))</code>)
'NHoodSize'	Vector de dos elementos de enteros positivos impares: [M N]. ' <code>NHoodSize</code> ' especifica el tamaño de la vecindad que se suprimirá, que es el conjunto de píxeles alrededor de cada pico que se pondrá a cero después de identificar el pico. (Por defecto: los valores impares más pequeños que sean mayores o iguales que <code>size(H)/50</code>)

`H` es la matriz devuelta por la función `hough`. `numpeaks` es un escalar entero positivo.

Ejemplos de uso

Ejemplo 1: Localizar y mostrar los picos en la Transformada Hough de una imagen.

```
I = imread('3punto.png');
BW=im2bw(I);
imshow(BW)
[H, T, R] = hough(BW);
P = houghpeaks(H);
figure, imshow(imadjust(mat2gray(H)), 'XData', T, 'YData', R,
'InitialMag', 'fit')
xlabel('\theta'), ylabel ('\rho');
axis on, axis normal, hold on;
plot(T(P(:,2)),R(P(:,1)),'s','color','red','LineWidth',4);
```

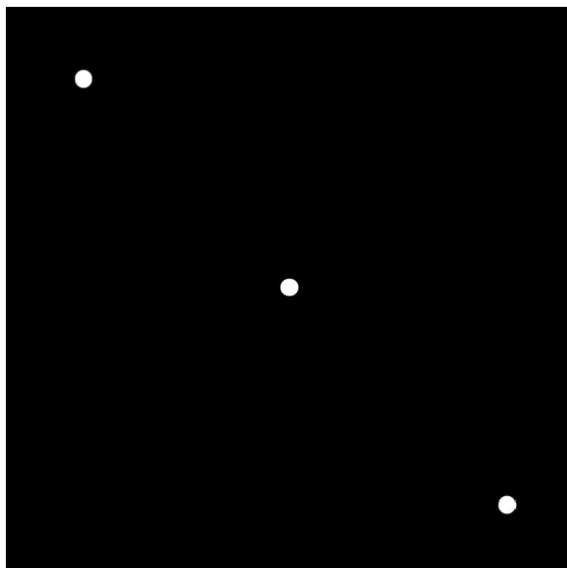


Figura f.207 - Imagen original (I)

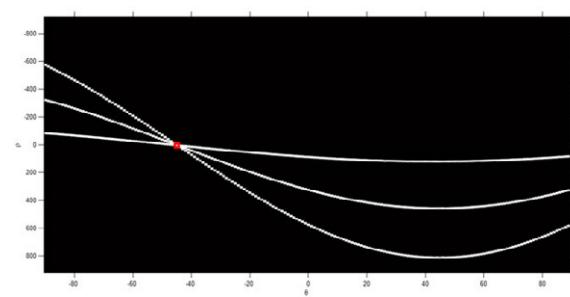


Figura f.218 - Picos detectados en la Transformada Hough de la imagen I

Ejemplo 2: Detectar los segmentos de línea en la imagen dada:

a) Directamente

```
RGB = imread('lineas.png');
imshow(RGB)
BW = im2bw(RGB);
[H,T,R] = hough(BW);

% En la función houghpeaks se suele especificar un número de
% máximos/picos mayor al número de líneas de la imagen.

P = houghpeaks(H,20)
lineas = houghlines(BW,T,R,P);

% Visualizar las líneas encontradas.

fh = figure;
imshow(BW), hold on
max_len = 0;
for k = 1:length(lineas)
    xy = [lineas(k).point1; lineas(k).point2];
    plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','red');
```

```

plot(xy(1,1),xy(1,2),'x','LineWidth',2,'Color','yellow');
plot(xy(2,1),xy(2,2),'x','LineWidth',2,'Color','green');
end
P =
    1353      139
    1530      140
    1208      140

```

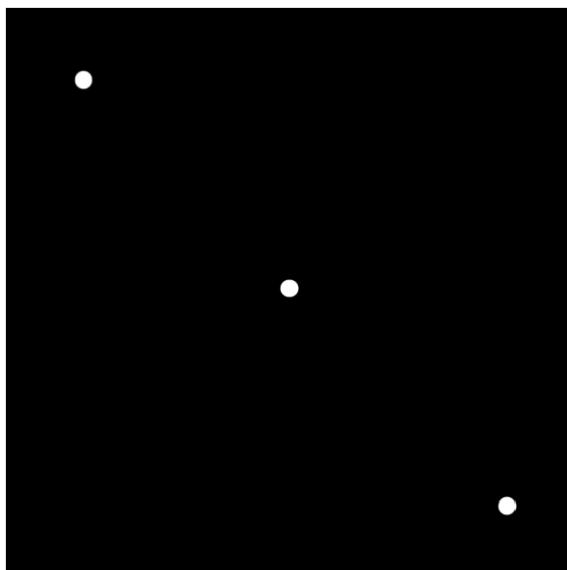


Figura f.207 - Imagen original (RGB)

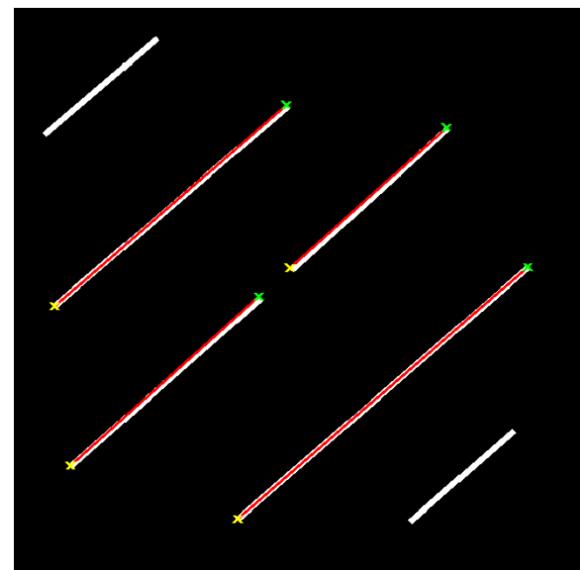


Figura f.219 - Segmentos detectados con los valores por defecto

b) Variando el umbral de la función houghpeaks

```

RGB = imread('lineas.png');
imshow(RGB)
BW = im2bw(RGB);
[H,T,R] = hough(BW);

% Variamos el umbral 'Threshold' de detección de la función houghpeaks
% para detecte más segmentos. Podríamos seleccionar luego las líneas de
% tamaño mayor usando el parámetro 'MinLength' de la función houghlines.

P = houghpeaks(H,20,'Threshold',0.5)
lineas = houghlines(BW,T,R,P);

```

% Visualizar las líneas encontradas.

```

fh = figure;
imshow(BW), hold on
max_len = 0;
for k = 1:length(lineas)
    xy = [lineas(k).point1; lineas(k).point2];
    plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','red');
    plot(xy(1,1),xy(1,2),'x','LineWidth',2,'Color','yellow');
    plot(xy(2,1),xy(2,2),'x','LineWidth',2,'Color','green');
end

```

```

P =
    1353      139
    1530      140
    1208      140

```

1362	136
1053	139
1052	142
1661	139
1360	142
1659	142
1340	136
1205	143
1661	136
1520	143
1521	136
1541	136
1198	136
1051	136
1340	143
1049	145
1220	136

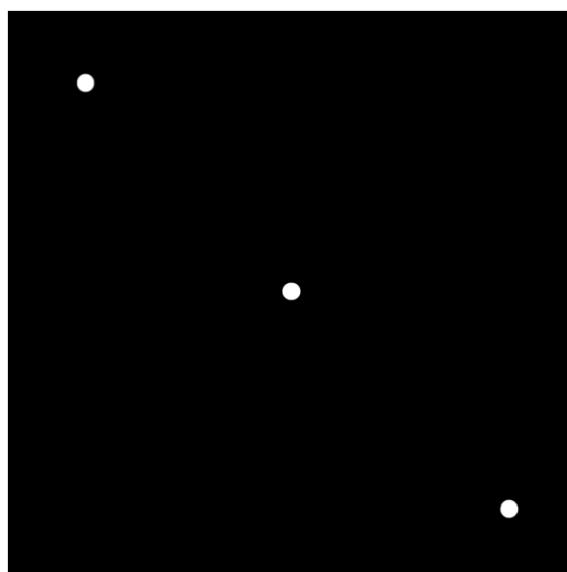


Figura f.207 - Imagen original (RGB)

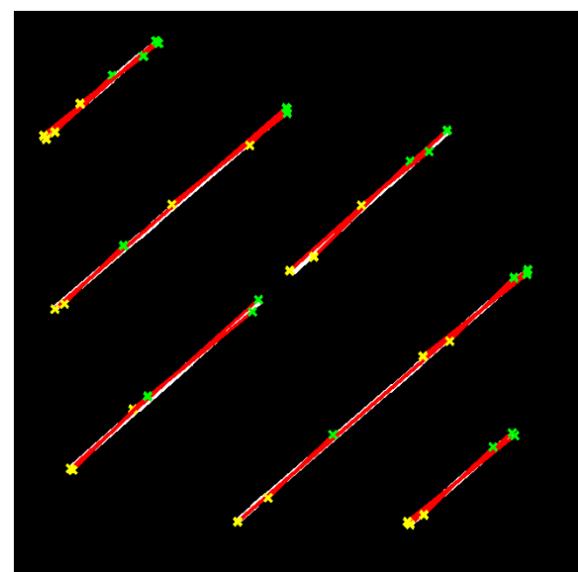


Figura f.220 - Segmentos detectados variando umbral de houghpeaks

c) Variando la resolución de theta y rho en la Transformada de Hough

```

RGB = imread('lineas.png');
imshow(RGB)
BW = im2bw(RGB);
[H,T,R] = hough(BW);

% Buscamos la resolución de theta y rho de la Transformada Hough donde
% houghpeaks detecte al menos 6 picos y la usamos.

v=0;
for i=0.1:.1:89
[H,T,R] = hough(BW,'RhoResolution',i,'ThetaResolution',i);

% Se define un valor superior a las líneas conocidas.

P = houghpeaks(H,20);
[a, b]=size(P);
if (a>=6)&&(v==0)
v=i;

```

```
end  
end  
  
[H,T,R] = hough(BW,'RhoResolution',v,'ThetaResolution',v);  
P = houghpeaks(H,20)  
lineas = houghlines(BW,T,R,P);  
  
% Visualizar las líneas encontradas.  
  
fh = figure;  
imshow(BW), hold on  
max_len = 0;  
for k = 1:length(lineas)  
xy = [lineas(k).point1; lineas(k).point2];  
plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','red');  
plot(xy(1,1),xy(1,2),'x','LineWidth',2,'Color','yellow');  
plot(xy(2,1),xy(2,2),'x','LineWidth',2,'Color','green');  
end  
  
P =  
13606 1351  
13394 1351  
16590 1351  
15169 1351  
15367 1351  
11959 1351  
12150 1351  
10531 1351  
13528 1376  
15280 1400
```

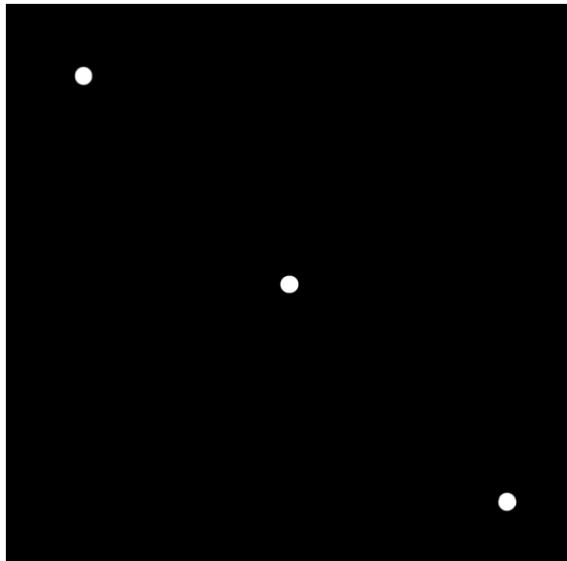


Figura f.207 - Imagen original (RGB)

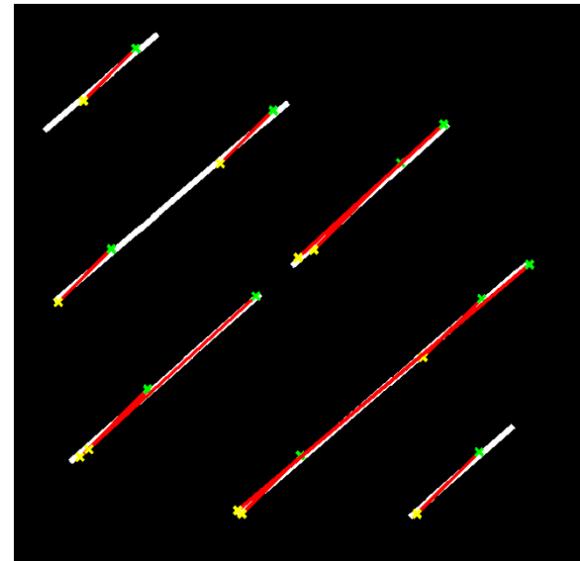


Figura f.221 - Segmentos detectados variando resolución de theta y rho de la Transformada Hough

Funciones relacionadas

hough, houghlines

iccfind

Búsqueda de perfiles ICC

Introducción

Dentro del ámbito de la Gestión del color, un perfil ICC es un conjunto de datos que caracteriza a un dispositivo de entrada o salida de color, o espacio color, según los estándares promulgados por el Consorcio Internacional del Color (ICC).^[21]

Sintaxis

```
P = iccfind(directory)
[P, descriptions] = iccfind(directory)
[...] = iccfind(directory, pattern)
```

Descripción

`P = iccfind(directory)` busca todos los perfiles ICC en el directorio especificado en la variable `directory`. La función devuelve `P`, un array de celdas con estructuras que contienen la información del perfil.

`[P, descriptions] = iccfind(directory)` busca todos los perfiles ICC en el directorio especificado y devuelve `P`, un array de celdas con estructuras que contienen la información del perfil, y `descriptions`, un array de celdas con cadenas de texto donde cada cadena describe el perfil correspondiente en la variable `P`. Cada cadena de texto es el valor del campo `Description.String` en la estructura de información del perfil.

`[...] = iccfind(directory, pattern)` devuelve todos los perfiles ICC del directorio especificado que tengan en su campo `Description.String` el patrón especificado en la variable `pattern`. `iccfind` busca sin tener en cuenta las mayúsculas.

Nota: Para mejorar el rendimiento, `iccfind` guarda temporalmente copias de los perfiles ICC en memoria. Puede que al añadir o modificar perfiles no se modifiquen los resultados de búsqueda. En ese caso se deben limpiar los archivos temporales usando el comando `clear`.

Ejemplos de uso

Ejemplo 1: Leer todos los perfiles ICC del directorio por defecto.

```
profiles = iccfind(iccroot);
```

Ejemplo 2: Listar todos los perfiles ICC con cadenas de texto que describan a cada perfil.

```
[profiles, descriptions] = iccfind(iccroot);
```

Ejemplo 3: Buscar los perfiles cuyo campo Description contenga la palabra rgb.

```
[profiles, descriptions] = iccfind(iccroot, 'rgb');
```

Funciones relacionadas

`iccread`, `iccroot`, `iccwrit`

iccread

Lectura de un perfil ICC

Introducción

Dentro del ámbito de la Gestión del color, un perfil ICC es un conjunto de datos que caracteriza a un dispositivo de entrada o salida de color, o espacio color, según los estándares promulgados por el Consorcio Internacional del Color (ICC). [21]

Sintaxis

```
P = iccread(filename)
```

Descripción

`P = iccread(filename)` lee la información del perfil ICC especificado en la variable `filename`. El archivo puede ser un perfil ICC o un archivo TIFF que contenga un perfil ICC incrustado. Para saber si un archivo TIFF contiene un perfil ICC se debe utilizar la función `imfinfo` para leer los detalles del archivo y ver si existe el campo `ICCPProfileOffset`. `iccread` busca el archivo en el directorio actual, en un directorio en la ruta de MATLAB o en el directorio devuelto por la función `icccroot`, en ese orden.

`iccread` devuelve la información del perfil en la estructura `P`, un array de 1-por-1 cuyos campos contienen las estructuras de datos (llamadas etiquetas o *tags*) definidas en la especificación ICC. `iccread` puede leer perfiles que se hayan generado con la Versión 2 (ICC.1:2001-04) o la Versión 4 (ICC.1:2001-12) de las especificaciones ICC. Para más información sobre perfiles ICC, visitar el sitio web oficial www.color.org.

Los perfiles ICC proporcionan sistemas de administración de color con la información necesaria para convertir datos de color entre espacios de color nativos a los dispositivos y espacios de color independientes a los dispositivos, llamados Espacios de conexión de Perfiles (PCS, *Profile Connection Space*). Con la función `makecform` se puede usar el perfil como fuente o destino para calcular transformaciones del espacio de color.

El número de campos en `P` depende de la clase del perfil y de las opciones que se eligieron al crear el perfil. `iccread` devuelve todas las etiquetas para un perfil dado, ya sean

públicas o privadas. Las etiquetas privadas y ciertas etiquetas públicas se mantienen como datos de tipo `uint8` codificados. La siguiente tabla muestra los campos que se encuentran en cualquier estructura de perfil generada por `iccread`, en el orden que aparecen en la estructura:

Campo	Tipo de Dato	Descripción
Header	Array de estructura de 1-por-1	Cabecera del perfil. Proporciona información general a cerca del perfil, como su clase, espacio de color y espacio de conexión (PCS).
TagTable	Array de celdas de n-por-3	Tabla de etiquetas del perfil.
Copyright	Cadena de texto	Información de copyright.
Description	Array de estructura de 1-por-1	El campo <code>String</code> de la estructura es una cadena de texto que describe el perfil.
MediaWhitepoint	Array tipo <code>double</code>	Valores <i>XYZ</i> triestímulos del punto blanco del dispositivo.
PrivateTags	Array de celdas de m-por-2	Contiene todas las etiquetas privadas o no definidas en las especificaciones ICC. Las firmas de las etiquetas están en la primera columna y el contenido de la etiqueta en la segunda. Notar que la función <code>iccread</code> deja el contenido de estas etiquetas en formato codificado de 8-bits sin signo.
Filename	Cadena de texto	Nombre del archivo que contiene el perfil.

Adicionalmente, la estructura leída `P` puede contener una o más de las siguientes transformadas:

- Transformada de tres componentes, basada en matrices: Una transformada simple que se usa normalmente para transformar entre los espacios de color RGB y *XYZ*. Si la transformada está presente, `P` contiene un campo llamado `MatTRC`.
- Transformada de N-componentes basada en tablas de consulta LUT: Una transformada que se usa para transformaciones entre espacios de color que tienen una relación más compleja. Este tipo de transformación se encuentra en cualquiera de los siguientes campos de `P`:

```
AToB0 BToA0 Preview0
AToB1 BToA1 Preview1
AToB2 BToA2 Preview2
AToB3 BToA3 Gamut
```

Ejemplos de uso

Ejemplo 1: Leer un perfil ICC que describe un monitor típico de ordenador. Determinar el espacio de color de la fuente.

```
% Leemos el perfil y mostramos la estructura de datos.
P = iccread('sRGB.icm')

% Mostramos el campo ColorSpace de la cabecera que nos indica el espacio
de color de la fuente.

P.Header.ColorSpace

P =
    Header: [1x1 struct]
    TagTable: {17x3 cell}
    Copyright: 'Copyright (c) 1999 Hewlett-Packard Company'
    Description: [1x1 struct]
    MediaWhitePoint: [0.9505 1 1.0891]
    MediaBlackPoint: [0 0 0]
    DeviceMfgDesc: [1x1 struct]
    DeviceModelDesc: [1x1 struct]
    ViewingCondDesc: [1x1 struct]
    ViewingConditions: [1x1 struct]
        Luminance: [76.0365 80 87.1246]
        Measurement: [1x36 uint8]
        Technology: [115 105 103 32 0 0 0 0 67 82 84 32]
        MatTRC: [1x1 struct]
    PrivateTags: {}
    Filename: 'sRGB.icm'

ans =
    RGB
```

Funciones relacionadas

`applycform`, `iccfind`, `iccroot`, `iccwrite`, `isicc`, `makecform`

iccroot

Búsqueda del directorio de almacenamiento por defecto de los perfiles ICC

Introducción

Dentro del ámbito de la Gestión del color, un perfil ICC es un conjunto de datos que caracteriza a un dispositivo de entrada o salida de color, o espacio color, según los estándares promulgados por el Consorcio Internacional del Color (ICC). ^[21]

Sintaxis

```
rootdir = iccroot
```

Descripción

`rootdir = iccroot` devuelve el directorio del sistema que contiene los perfiles ICC. Se pueden guardar perfiles adicionales en otros directorios, pero esta es la localización por defecto que usa el sistema de administración de color.

Nota: Solo funciona en los sistemas operativos Windows y Mac OS X.

Ejemplos de uso

Ejemplo 1: Informar de todos los perfiles que se encuentran en el directorio por defecto.

```
iccfind(iccroot)
```

Funciones relacionadas

```
iccfind, iccread, iccwrite
```

iccwrite

Escritura de un perfil de color ICC a un archivo

Introducción

Dentro del ámbito de la Gestión del color, un perfil ICC es un conjunto de datos que caracteriza a un dispositivo de entrada o salida de color, o espacio color, según los estándares promulgados por el Consorcio Internacional del Color (ICC).^[21]

Sintaxis

```
P_new = iccwrite(P, filename)
```

Descripción

`P_new = iccwrite(P, filename)` escribe los datos del perfil de color ICC de la estructura `P` en el archivo especificado en la variable `filename`.

`P` es una estructura que representa un perfil ICC en el formato devuelto por la función `iccread` y utilizado por `makecform` y `applycform` para calcular transformaciones de espacio de color. `P` debe contener todas las etiquetas y campos que dictan las especificaciones del perfil ICC. Algunos campos pueden ser inconsistentes debido a cambios en la estructura, por ejemplo la tabla de etiquetas puede que no sea correcta porque las etiquetas se pueden haber añadido, borrado o modificado desde que se creó la tabla. `iccwrite` realiza las correcciones necesarias en la estructura antes de escribirla en el archivo y devuelve la estructura corregida en la variable `P_new`.

Nota: ICC recomienda modificar el campo `Description` del perfil ICC antes de escribir los datos en un archivo ya que puede haber sido modificado por alguna aplicación. Cada perfil debe tener una descripción única en ese campo.

`iccwrite` puede escribir los datos del perfil de color utilizando la Versión 2 (ICC.1:2001-04) o la Versión 4 (ICC.1:2001-12) de las especificaciones de ICC en función del valor de la variable `version` de la cabecera del perfil. `iccwrite` dará error si falta algún campo necesario. Para más información sobre perfiles ICC, visitar el sitio web oficial de ICC, www.color.org.

Nota: `iccwrite` no convierte automáticamente perfiles entre una versión y otra. Estas conversiones se deben hacer manualmente, añadiendo o modificando los campos necesarios. Usar la función `isicc` para validar un perfil.

Ejemplos de uso

Ejemplo 1: Leer un perfil en el espacio de trabajo y exportarlo a un nuevo archivo cambiando la descripción del perfil antes de escribir los datos.

```
P = iccread('monitor.icm');
P.Description.String = 'Mi nueva Descripción';
pmon = iccwrite(P, 'monitor2.icm');
```

Funciones relacionadas

`applycform`, `iccread`, `isicc`, `makecform`

idct2

Transformada de coseno discreta inversa en 2-D

Introducción

La Transformada de Coseno Discreta (DCT) representa una imagen como una suma de sinusoides de magnitudes y frecuencias oscilantes. La DCT tiene la propiedad de que para una imagen corriente, la mayoría de información o energía importante para su visualización se localiza solo en unos pocos coeficientes de la DCT. Por eso es la transformada más utilizada en los estándares de compresión de imágenes. Por ejemplo, es la transformada que se usa en la compresión JPEG, Mp3 o MPEG.

La Transformada de coseno discreta inversa (IDCT o DCT-III) reconstruye una secuencia de datos a partir de los coeficientes DCT actuando como decodificador.

Sintaxis

```
B = idct2(A)
B = idct2(A,m,n)
B = idct2(A,[m n])
```

Descripción

`B = idct2(A)` devuelve la Transformada de coseno discreta inversa en dos dimensiones de `A`.

`B = idct2(A,m,n)` rellena `A` con 0's hasta alcanzar el tamaño `m`-por-`n` antes de realizar la transformada. Si `[m n] < size(A)`, `idct2` corta `A` antes de realizar la transformada.

`B = idct2(A,[m n])` igual que la expresión anterior.

Para cualquier `A`, `idct2(dct2(A))` es igual (salvo errores de redondeo).

La matriz de entrada `A` puede ser de tipo `double` o de cualquier clase numérica. La matriz de salida `B` es de tipo `double`.

Ejemplos de uso

Ejemplo 1: Calcular la transformada de coseno discreta de una imagen RGB y comprimir la imagen eliminando la información redundante con cuantificación uniforme escalar de valor q=50. Recuperar la imagen con la transformada inversa.

```
% Leemos una imagen RGB y procesamos cada color por separado.

RGB = imread('autumn.jpg');
imshow(RGB)
DIR = dir('autumn.jpg');
tamorig=DIR.bytes;

% DCT de cada color.

JR=dct2(RGB(:,:,1));
JG=dct2(RGB(:,:,2));
JB=dct2(RGB(:,:,3));

% Cuantificación uniforme (compresión) de cada color.

q = 50;
qR=round(JR/q);
qG=round(JG/q);
qB=round(JB/q);

% Cuantificación inversa de cada color.

iqR=qR.*q;
iqG=qG.*q;
iqB=qB.*q;

% IDCT de cada color.

JR=idct2(iqR);
JG=idct2(iqG);
JB=idct2(iqB);

% Creamos la imagen final comprimida, mostramos y comparamos el tamaño.

J(:,:,1)=JR;
J(:,:,2)=JG;
J(:,:,3)=JB;
J = uint8(J);
figure, imshow(J)

tamorig
imwrite (J, 'pautumn.jpg');
DIR = dir('pautumn.jpg');
tamproc=DIR.bytes

tamorig =
    1266731

tamproc =
    237376
```



Figura f.107 - Imagen original (RGB)



Figura f.222 - Imagen comprimida J

Funciones relacionadas

`dct2, dctmtx, fft2, ifft2`

ifanbeam

Transformada en abanico inversa

Sintaxis

```
I = ifanbeam(F,D)
I = ifanbeam(...,param1,val1,param2,val2,...)
[I,H] = ifanbeam(...)
```

Descripción

`I = ifanbeam(F,D)` reconstruye la imagen `I` a partir de las proyecciones en abanico en el array bidimensional `F`. Cada columna de `F` contiene datos de proyección de haz en abanico a un ángulo de rotación determinado. `ifanbeam` asume que el centro de rotación es el punto central de las proyecciones, que se define como `ceil(size(F,1)/2)`, y que los ángulos de rotación de entrada vienen espaciados con igual incremento cubriendo el rango `[0:359]` grados. El incremento será el mismo que tendrán los ángulos de medida o posiciones de los sensores.

`D` es la distancia desde el vértice del haz en abanico al centro de la rotación.

`I = ifanbeam(...,param1,val1,param2,val2,...)` especifica parámetros que controlan varios aspectos de la reconstrucción descritos en la siguiente tabla. Los nombres de los parámetros se pueden abbreviar y no son sensibles a las mayúsculas. Para más información consultar la página de la función `fanbeam`.

Parámetro	Descripción
'FanCoverage'	Cadena que especifica la cobertura de rotación de los ejes: 'cycle' — Rotar completamente [0,360). (Por defecto) 'minimal' — Rotar lo mínimo necesario para representar el objeto.
'FanRotationIncrement'	Escalar real positivo que especifica el incremento del ángulo de rotación de las proyecciones de haz en abanico. Medido en grados. (Por defecto: 1)
'FanSensorGeometry'	Cadena que especifica cómo se posicionan los sensores: 'arc' — Los sensores están equiespaciados a través de un arco circular a una distancia <code>D</code> del centro de rotación. (Por defecto)

	<p>Figura f.161 - Sensores en arco</p>
	<p>'line' — Los sensores están equiespaciados a lo largo de una línea.</p> <p>Figura f.162 - Sensores en línea</p>
'FanSensorSpacing'	<p>Escalar real positivo que especifica el espaciado de los sensores de haz en abanico. La interpretación del valor depende de la variable 'FanSensorGeometry' .</p> <p>Si 'FanSensorGeometry' se pone a 'arc' (por defecto) el valor define el espaciado angular en grados (por defecto: 1)</p> <p>Si 'FanSensorGeometry' vale 'line', el valor define el espaciado lineal (por defecto: 1). Consultar la función fanbeam para más detalle.</p> <p>Nota: Este espaciado lineal se mide en el eje x. El origen de los ejes es el pixel central de la imagen.</p>
'Filter'	Cadena que especifica el nombre de un filtro. Consultar la función iradon para más detalles.
'FrequencyScaling'	Escalar en el rango (0, 1] que modifica el filtro reescalando su eje de frecuencias. Consultar la función iradon para más detalles.
'Interpolación'	Cadena de texto que especifica el tipo de interpolación que

	se usará entre los datos de haz paralelo y haz en abanico. 'nearest' — Vecino más cercano {'linear'} — Lineal 'spline' — spline cúbico a trozos 'pchip' — Hermite (PCHIP) cúbico a trozos 'cubic' — Igual que 'pchip'
'OutputSize'	Escalar positivo que especifica el número de filas y columnas en la imagen reconstruida. Si no se especifica 'OutputSize', ifanbeam determina el tamaño automáticamente. Si se especifica 'OutputSize', ifanbeam reconstruye una porción más corta o más larga de la imagen, pero no cambia la escala de los datos. Nota: Si las proyecciones se calcularon con la función fanbeam, la imagen reconstruida puede no tener el mismo tamaño que la imagen original.

[I, H] = ifanbeam(. . .) devuelve la respuesta en frecuencia del filtro especificado en el vector H.

Los parámetros de entrada F y D pueden ser de tipo double o single. El resto de entradas numéricas deben ser de tipo double. Los argumentos de salida son tipo double.

Ejemplos de uso

Ejemplo 1: Crear una transformación de haz en abanico de una imagen “cabeza de práctica” (*phantom head*) y recuperar la imagen a través de las proyecciones.

```

ph = phantom(512);
d = 400;

% Proyecciones en abanico
F = fanbeam(ph,d);

% Transformada en abanico inversa
phr = ifanbeam(F,d);

imshow(ph),
figure, imshow(phr)

```



Figura f.165 - Imagen original (ph)

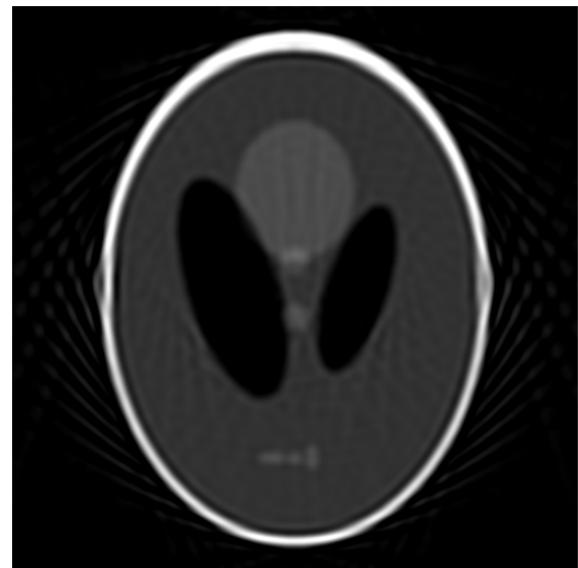


Figura f.223 - Imagen recuperada (phr)

Ejemplo 2: Repetir el ejercicio anterior usando el parámetro de cobertura mínima.

% Debemos utilizar el parámetro 'fancovariance' puesto a 'minimal' tanto en la creación de las proyecciones en abanico como en la transformada en abanico inversa. Para ello debemos usar la Transformada de Radón para sacar las proyecciones paralelas y convertirlas luego a abanico utilizando el parámetro especificado:

```
ph = phantom(512);
P = radon(ph);
[F,obeta,otheta] = para2fan(P,400,...  

    'FanSensorSpacing',0.5,...  

    'FanCoverage','minimal',...  

    'FanRotationIncrement',1);  

% Reconstruimos.  

phr = ifanbeam(F,400,...  

    'FanSensorSpacing',0.5,...  

    'Filter','Shepp-Logan',...  

    'OutputSize',512,...  

    'FanCoverage','minimal',...  

    'FanRotationIncrement',1);  

imshow(ph)
figure, imshow(phr)
```



Figura f.127 - Imagen original (ph)

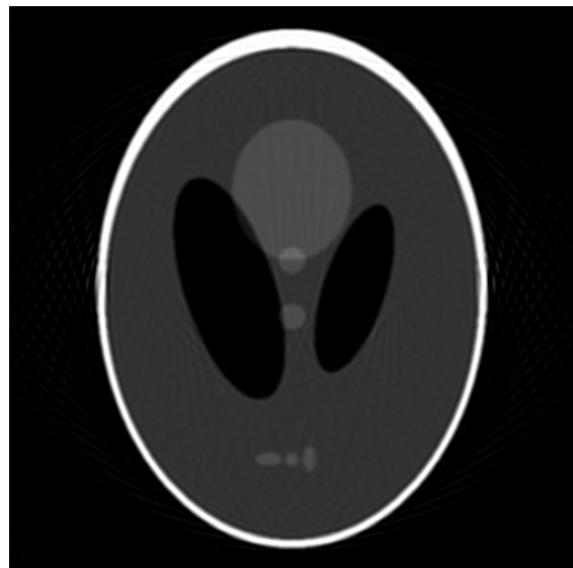


Figura f.224 - Imagen recuperada (phr)

Funciones relacionadas

`fan2para`, `fanbeam`, `iradon`, `para2fan`, `phantom`, `radon`

im2bw

Conversión a imagen binaria mediante un umbral

Sintaxis

```
BW = im2bw(I, level)
BW = im2bw(X, map, level)
BW = im2bw(RGB, level)
```

Descripción

`BW = im2bw(I, level)` convierte a binario la imagen `I` en escala de grises. La imagen de salida `BW` tendrá valor 1 (blanco) en todos los píxeles de la imagen de entrada que tengan una luminancia mayor que el nivel o umbral especificado en la variable `level`, y valor 0 (negro) en el resto de píxeles. El umbral de la variable `level` será del rango [0, 1] y es relativo a los niveles de señal posibles para el tipo/clase de imagen. Por ejemplo, un valor de 0.5 significa que está en medio del blanco y el negro, sin tener en cuenta el tipo/clase de la imagen. Para calcular el umbral se puede utilizar la función `graythresh` que devuelve el umbral óptimo según el método de Otsu. Si no se especifica el umbral en la variable `level`, `im2bw` utiliza el valor de 0.5.

`BW = im2bw(X, map, level)` convierte la imagen indexada `X` a imagen binaria, utilizando el mapa de color de la variable `map`.

`BW = im2bw(RGB, level)` convierte la imagen `RGB` en color verdadero a imagen binaria.

Si la imagen de entrada no es una imagen en escala de grises, la función `im2bw` convierte la imagen de entrada a escala de grises y después la convierte a binaria mediante umbralización.

La imagen de entrada puede ser de tipo `uint8`, `uint16`, `single`, `int16`, o `double` y debe ser de tipo no disperso. La imagen de salida `BW` es de tipo `logical`. `I` y `X` deben ser bidimensionales. Las imágenes `RGB` son de tamaño `M`-por-`N`-por-3.

Ejemplos de uso

Ejemplo 1: Convertir una imagen RGB a binario utilizando el umbral óptimo devuelto por la función `graythresh`.

```
I = imread('paints.tif');
imshow(I)
threshold = graythresh(I);
BW = im2bw(I,threshold);
figure, imshow(BW)
```



Figura f.225 - Imagen original (I)



Figura f.226 - Imagen convertida a binaria (BW) con
im2bw

Ejemplo 2: Umbralizar la imagen variando el umbral hasta conseguir separar la imagen principal del fondo.

```
I = imread('bird.tif');
imshow(I)

% Empezamos utilizando el umbral óptimo.

threshold = graythresh(I);
BW = im2bw(I,threshold);
figure, imshow(BW)

% El umbral óptimo no funciona bien con esta imagen. Probamos valores de
% umbral hasta conseguir el efecto buscado.

threshold = 0.2;
BW2 = im2bw(I,threshold);
figure, imshow(BW)
threshold = 0.4;
BW3 = im2bw(I,threshold);
figure, imshow(BW)
```

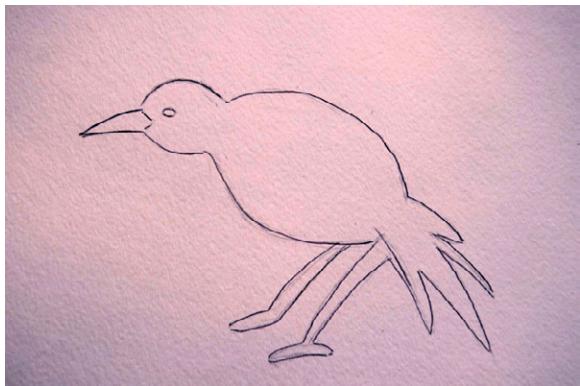
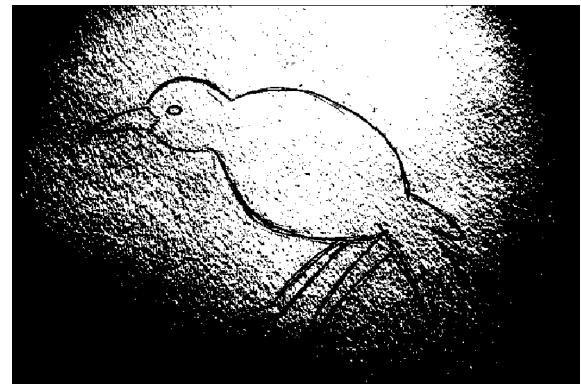
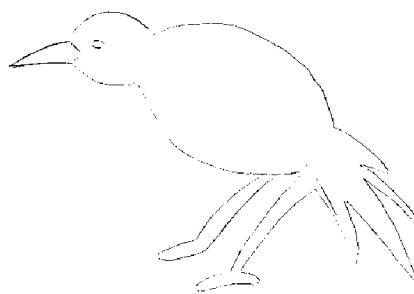


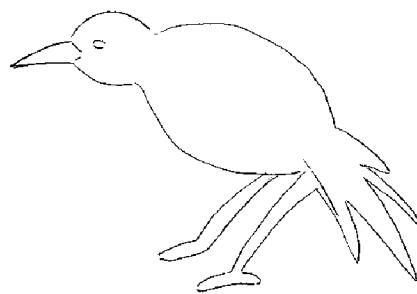
Figura f.227 - Imagen original (I)



*Figura f.228 - Imagen convertida a binaria (BW)
usando umbral óptimo*



*Figura f.229 - Imagen convertida a binaria (BW2)
usando umbral de 0.2*



*Figura f.230 - Imagen convertida a binaria (BW3)
usando umbral de 0.4*

Funciones relacionadas

`graythresh`, `ind2gray`, `rgb2gray`

im2col

Recolocación de bloques de imagen en columnas

Introducción

La función `im2col` reordena bloques de una imagen en columnas, mientras que `col2im` realiza la operación complementaria. El uso de estas funciones es bastante confuso por lo que se suele intentar evitarlas utilizando otras funciones más avanzadas y específicas como `reshape`, `blockproc` o `nlfiltter`.

Sintaxis

```
B = im2col(A,[m n],block_type)
B = im2col(A,'indexed',...)
```

Descripción

`B = im2col(A,[m n],block_type)` recoloca en columnas los bloques de tamaño m -por- n de una imagen. La variable `block_type` determina la forma como se cogen los bloques y puede tomar cualquiera de los siguientes valores:

Valor	Descripción
'distinct'	Recoloca cada bloque distinto de tamaño m -por- n de la imagen <code>A</code> en una columna de <code>B</code> . <code>im2col</code> llena <code>A</code> con ceros si es necesario, para que su tamaño sea un entero múltiplo de m -por- n . Si <code>A = [A11 A12; A21 A22]</code> , donde cada A_{ij} es de tamaño m -por- n , entonces <code>B = [A11(:) A12(:) A21(:) A22(:)]</code> .
'sliding'	Reordena cada bloque deslizante de <code>A</code> de tamaño m -por- n (los bloques van barriendo el array <code>A</code>) en una columna de <code>B</code> , sin llenar con ceros, así que solo se cogen los bloques que puedan construirse con una vecindad completa de m -por- n (puede producir una imagen resultante de tamaño menor a la original). <code>B</code> tiene $m \times n$ filas y tantas columnas como vecindades de m -por- n de <code>A</code> haya. Si el tamaño de <code>A</code> es $[mm \ nn]$ entonces el de <code>B</code> será de $(m \times n)$ -por- $((mm-m+1) \times (nn-n+1))$. Cada columna de <code>B</code> contendrá los vecinos de <code>A</code> reordenados como <code>NHOOD(:)</code> donde <code>NHOOD</code> es una matriz que contiene una vecindad de m -por- n de <code>A</code> . <code>im2col</code> ordena las columnas de <code>B</code> para que se puedan recolocar posteriormente para formar una matriz normal.

	<p>Esta forma está pensada para operaciones sobre las columnas de B, por ejemplo utilizando la función <code>sum(B)</code> que devuelve un escalar para cada columna de B (vecindad de A), realizando una suma alrededor de esa vecindad y pudiendo guardar directamente el resultado en una matriz de tamaño $(mm-m+1) \times (nn-n+1)$ que es lo que la función complementaria <code>col2im</code> espera recibir para reconstruir la imagen después del procesado.</p> <pre>B = im2col(A,[m n],'sliding'); C = col2im(sum(B),mm-m+1,nn-n+1,'sliding');</pre> <p>(Por defecto)</p>
--	---

`B = im2col(A, 'indexed', ...)` procesa **A** como una imagen indexada, rellenando con ceros si la clase de **A** es `uint8`, o con unos si la clase es `double`.

La imagen de entrada **A** puede ser de tipo `numeric` o `logical`. La matriz de salida **B** es del mismo tipo/clase que la imagen de entrada. Los bloques se cogen de izquierda a derecha y de arriba abajo en **A** y las columnas se añaden de izquierda a derecha en **B**.

Ejemplos de uso

Ejemplo 1: Comparar las opciones '`sliding`' y '`distinct`' a la hora de recolocar los bloques de datos con `im2col`.

```
A = [1:10;11:20;21:30;31:40]
S = im2col(A,[4,3],'sliding')
D = im2col(A,[4,3],'distinct')
```

```
A =
    1     2     3     4     5     6     7     8     9    10
   11    12    13    14    15    16    17    18    19    20
   21    22    23    24    25    26    27    28    29    30
   31    32    33    34    35    36    37    38    39    40

S =
    1     2     3     4     5     6     7     8
   11    12    13    14    15    16    17    18
   21    22    23    24    25    26    27    28
   31    32    33    34    35    36    37    38
    2     3     4     5     6     7     8     9
   12    13    14    15    16    17    18    19
   22    23    24    25    26    27    28    29
   32    33    34    35    36    37    38    39
    3     4     5     6     7     8     9    10
   13    14    15    16    17    18    19    20
   23    24    25    26    27    28    29    30
   33    34    35    36    37    38    39    40
```

```
D =
    1     4     7    10
   11    14    17    20
   21    24    27    30
   31    34    37    40
    2     5     8     0
   12    15    18     0
   22    25    28     0
   32    35    38     0
    3     6     9     0
   13    16    19     0
   23    26    29     0
   33    36    39     0
```

Ejemplo 2: Pasar cada componente de una imagen RGB a una matriz columna.

```
RGB = imread('rgb.jpg')
rc = im2col(RGB(:,:,1), [size(RGB,1) size(RGB,2)]);
gc = im2col(RGB(:,:,2), [size(RGB,1) size(RGB,2)]);
bc = im2col(RGB(:,:,3), [size(RGB,1) size(RGB,2)]);
```

Ejemplo 3: Reorganizar un array en columnas con la función `im2col` y recuperar el array original con la función inversa.

```
A = [1:10;11:20;21:30;31:40]
B = im2col(A,[2,5],'distinct')
C = col2im(B,[2,5],[4,10],'distinct')
```

```
A =
    1     2     3     4     5      6     7     8     9    10
   11    12    13    14    15     16    17    18    19    20
   21    22    23    24    25     26    27    28    29    30
   31    32    33    34    35     36    37    38    39    40

B =
    1     21     6     26
   11    31    16     36
    2     22     7     27
   12    32    17     37
    3     23     8     28
   13    33    18     38
    4     24     9     29
   14    34    19     39
    5     25    10     30
   15    35    20     40

C =
    1     2     3     4     5      6     7     8     9    10
   11    12    13    14    15     16    17    18    19    20
   21    22    23    24    25     26    27    28    29    30
   31    32    33    34    35     36    37    38    39    40
```

Funciones relacionadas

`blockproc`, `col2im`, `colfilt`, `nlfilt`

im2double

Conversión de una imagen a clase double

Introducción

MATLAB guarda y opera por defecto con datos de tipo `double`. Cada dato del array se guarda con una precisión de 8 bytes, lo que no es muy eficiente al trabajar con imágenes de muchos píxeles. De ahí que se suela usar la clase `uint8` en procesado de imágenes ya que requiere solo un byte por elemento y tiene un rango de 0 a 255, siendo 8 veces más eficiente que el tipo `double`. Para evitar problemas al trabajar con las funciones predefinidas de la *toolbox* de procesado de imágenes, conviene trabajar con datos en formato `double` y cerciorarse de que los datos están escalados al rango [0, 1].

MATLAB dispone de dos funciones para la conversión de imágenes a tipo `double`. `im2double`, que convierte a un array tipo `double` escalando en el rango [0, 1] y `double`, que realiza la conversión sin reescalar.

Sintaxis

```
I2 = im2double(I)
RGB2 = im2double(RGB)
I = im2double(BW)
X2 = im2double(X,'indexed')
```

Descripción

`I2 = im2double(I)` convierte la imagen de intensidades `I` a clase `double`, reescalando los datos si es necesario. Si la imagen de entrada es de clase `double`, la de salida será idéntica.

`RGB2 = im2double(RGB)` convierte la imagen en color verdadero `RGB` a clase `double`, reescalando los datos si es necesario.

`I = im2double(BW)` convierte la imagen binaria `BW` a una imagen de intensidades de clase `double`.

`X2 = im2double(X,'indexed')` convierte la imagen indexada `X` a clase `double`, compensando los datos si es necesario.

Las imágenes de intensidad y de color verdadero pueden ser de clase `uint8`, `uint16`, `double`, `logical`, `single`, o `int16`. Las imágenes indexadas pueden ser de clase `uint8`, `uint16`, `double` o `logical`. Las imágenes binarias de entrada deben ser de clase `logical`. La imagen de salida es de clase `double`.

Ejemplos de uso

Ejemplo 1: Convertir una matriz imagen de clase `uint8` a clase `double`.

```
I1 = reshape(uint8(linspace(1,255,9)),[3 3])
I2 = im2double(I1)

I1 =
    1    96   192
   33   128   223
   65   160   255

I2 =
    0.0039    0.3765    0.7529
    0.1294    0.5020    0.8745
    0.2549    0.6275    1.0000
```

Funciones relacionadas

`double`, `im2single`, `im2int16`, `im2uint8`, `im2uint16`

im2int16

Conversión de una imagen a clase int16 (enteros con signo de 16-bits)

Sintaxis

```
I2 = im2int16(I)
RGB2 = im2int16(RGB)
I = im2int16(BW)
```

Descripción

`I2 = im2int16(I)` convierte la imagen de intensidades `I` a clase `int16`, reescalando los datos si es necesario. Si la imagen de entrada es de clase `int16`, la de salida será idéntica.

`RGB2 = im2int16(RGB)` convierte la imagen de color verdadero `RGB` a clase `int16`, reescalando los datos si es necesario.

`I = im2int16(BW)` convierte la imagen binaria `BW` a imagen binaria de intensidades de clase `int16` cambiando los elementos de valor cero o falso a -32768 y los de valor uno o verdadero a 32767.

Las imágenes de intensidad y de color verdadero pueden ser de clase `uint8`, `uint16`, `single`, `double` o `logical`. Las imágenes binarias de entrada deben ser de clase `logical`. La imagen de salida es de clase `int16`.

Ejemplos de uso

Ejemplo 1: Convertir una matriz imagen de clase `double` a clase `int16`.

```
I = reshape(linspace(0,1,6),[2 3])
I2 = im2int16(I)

I =
    0      0.4000      0.8000
    0.2000     0.6000      1.0000

I2 =
   -32768   -6554   19660
  -19661    6553   32767
```

Funciones relacionadas

`im2double`, `im2single`, `im2uint8`, `im2uint16`, `int16`

im2java2d

Conversión de una imagen a tipo *bufferedimage* de Java

Sintaxis

```
jimage = im2java2d(I)
jimage = im2java2d(X,MAP)
```

Descripción

`jimage = im2java2d(I)` convierte la imagen `I` a una instancia de la clase de imagen de Java `java.awt.image.BufferedImage`. La imagen `I` puede ser de intensidades (escala de grises), RGB o binaria.

`jimage = im2java2d(X,MAP)` convierte la imagen indexada `X` con mapa de color `MAP` a una instancia de la clase de imagen de Java `java.awt.image.BufferedImage`.

Nota: La función `im2java2d` trabaja con el API de Java en 2D. La función `im2java` trabaja con el kit de herramientas de ventana abstracta (AWT, *Abstract Windowing Toolkit*).

Las imágenes de entrada de intensidades, indexadas y RGB pueden ser de clase `uint8`, `uint16` o `double`. Las imágenes binarias de entrada deben ser de clase `logical`.

Ejemplos de uso

Ejemplo 1: Leer una imagen en MATLAB y utilizar `im2java2d` para convertirla en una instancia de la clase de imagen de Java `java.awt.image.BufferedImage` y mostrarla en una ventana de Java.

```
% Convertimos la imagen.

I = imread('rueda.tif');
javaImage = im2java2d(I);

% La mostramos como ventana de Java.

frame = javax.swing.JFrame;
icon = javax.swing.ImageIcon(javaImage);
label = javax.swing.JLabel(icon);
frame.getContentPane.add(label);
frame.pack
frame.show
```

im2single

Conversión de una imagen a clase single

Sintaxis

```
I2 = im2single(I)
RGB2 = im2single(RGB)
I = im2single(BW)
X2 = im2single(X, 'indexed')
```

Descripción

`I2 = im2single(I)` convierte la imagen de intensidades `I` a clase `single`, reescalando los datos si es necesario. Si la imagen de entrada es de clase `single`, la imagen de salida será idéntica.

`RGB2 = im2single(RGB)` convierte la imagen de color verdadero `RGB` a clase `single`, reescalando los datos si es necesario.

`I = im2single(BW)` convierte la imagen binaria `BW` a una imagen de intensidades de clase `single`.

`X2 = im2single(X, 'indexed')` convierte la imagen indexada `X` a una imagen de clase `single`, compensando los datos si es necesario.

Las imágenes de intensidades y de color verdadero pueden ser de clase `uint8`, `uint16`, `int16`, `single`, `double` o `logical`. Las imágenes indexadas pueden ser de clase `uint8`, `uint16`, `double` o `logical`. Las imágenes binarias de entrada deben ser de clase `logical`. La imagen de salida es de clase `single`.

Ejemplos de uso

Ejemplo 1: Convertir una matriz imagen de clase `uint8` a clase `single`.

```
I = reshape(uint8(linspace(1,255,9)),[3 3])
I2 = im2single(I)
```

```
I =
    1    96   192
   33   128   223
   65   160   255
```

```
I2 =  
0.0039    0.3765    0.7529  
0.1294    0.5020    0.8745  
0.2549    0.6275    1.0000
```

Funciones relacionadas

`im2double`, `im2int16`, `im2uint8`, `im2uint16`, `single`

im2uint16

Conversión de una imagen a clase uint16

Sintaxis

```
I2 = im2uint16(I)
RGB2 = im2uint16(RGB)
I = im2uint16(BW)
X2 = im2uint16(X, 'indexed')
```

Descripción

`I2 = im2uint16(I)` convierte la imagen de intensidades `I` a clase `uint16` (enteros sin signo de 16-bits), reescalando los datos si es necesario. Si la imagen de entrada es de clase `uint16`, la imagen de salida será idéntica.

`RGB2 = im2uint16(RGB)` convierte la imagen en color verdadero `RGB` a clase `uint16`, reescalando los datos si es necesario.

`I = im2uint16(BW)` convierte la imagen binaria `BW` a imagen de intensidades de clase `uint16`, poniendo los elementos de valor 1 a valor 65535.

`X2 = im2uint16(X, 'indexed')` convierte la imagen indexada `X` a clase `uint16`, compensando los datos si es necesario. Si `X` es de clase `double`, `max(X(:))` debe valer 65536 o menos.

Las imágenes de intensidades y de color verdadero pueden ser de clase `uint8`, `uint16`, `double`, `logical`, `single` o `int16`. Las imágenes indexadas pueden ser de clase `uint8`, `uint16`, `double` o `logical`. Las imágenes binarias de entrada deben ser de clase `logical`. La imagen de salida es de clase `uint16`.

Ejemplos de uso

Ejemplo 1: Convertir una matriz imagen de clase `double` a clase `uint16`.

```
I = reshape(linspace(0,1,9),[3 3])
I2 = im2uint16(I)

I =
    0      0.3750      0.7500
   0.1250      0.5000      0.8750
```

0.2500 0.6250 1.0000

I2 =

0	24576	49151
8192	32768	57343
16384	40959	65535

Funciones relacionadas

[im2uint8](#), [double](#), [im2double](#), [uint8](#), [uint16](#), [imapprox](#)

im2uint8

Conversión de una imagen a clase uint8

Sintaxis

```
I2 = im2uint8(I1)
RGB2 = im2uint8(RGB1)
I = im2uint8(BW)
X2 = im2uint8(X1, 'indexed')
```

Descripción

`im2uint8` convierte una imagen de entrada a clase `uint8` (enteros sin signo de 8-bits). Si la imagen de entrada es de clase `uint8`, la imagen de salida será idéntica. Si la imagen de entrada no es de clase `uint8`, `im2uint8` devuelve una imagen equivalente de clase `uint8`, reescalando o compensando los datos si es necesario.

`I2 = im2uint8(I1)` convierte la imagen en escala de grises `I1` a clase `uint8`, reescalando los datos si es necesario.

`RGB2 = im2uint8(RGB1)` convierte la imagen en color verdadero `RGB1` a clase `uint8`, reescalando los datos si es necesario.

`I = im2uint8(BW)` convierte la imagen binaria `BW` a una imagen en escala de grises de clase `uint8`, cambiando los elementos de valor 1 por el valor 255.

`X2 = im2uint8(X1, 'indexed')` convierte la imagen indexada `X1` a clase `uint8`, compensando los datos si es necesario. Notar que no siempre es posible convertir una imagen indexada a clase `uint8`. Si `X1` es de clase `double`, el valor máximo de `X1` debe ser 256 o menos; si `X1` es de clase `uint16`, el valor máximo de `X1` debe ser 255 o menos.

Las imágenes en escala de grises y de color verdadero pueden ser de tipo `uint8`, `uint16`, `int16`, `single`, `double` o `logical`. Las imágenes indexadas pueden ser del tipo `uint8`, `uint16`, `double` o `logical`. Las imágenes binarias de entrada deben ser de clase `logical`. La imagen de salida es de clase `uint8`.

Ejemplos de uso

Ejemplo 1: Convertir una matriz imagen de clase uint16 a clase uint8.

```
I1 = reshape(uint16(linspace(0,65535,9)),[3 3])
I2 = im2uint8(I1)

I1 =
    0    24576   49151
    8192   32768   57343
   16384   40959   65535

I2 =
     0     96    191
    32    128    223
    64    159    255
```

Funciones relacionadas

[im2double](#), [im2int16](#), [im2single](#), [im2uint16](#), [uint8](#)

imabsdiff

Diferencia en valor absoluto de dos imágenes

Sintaxis

```
Z = imabsdiff(X,Y)
```

Descripción

`Z = imabsdiff(X,Y)` resta cada elemento del array `Y` al elemento correspondiente en el array `X` y devuelve la diferencia en valor absoluto en el elemento del array `Z`. `X` e `Y` son arrays numéricos reales y no dispersos de la misma clase y tamaño. `Z` será de la misma clase y tamaño que `X` e `Y`. Si `X` e `Y` son arrays enteros se truncarán los elementos a la salida que excedan el rango de los valores enteros.

Si `X` e `Y` son arrays de clase `double` se puede usar la expresión `abs(X-Y)` en vez de esta función.

Ejemplos de uso

Ejemplo 1: Calcular la diferencia en valor absoluto de dos arrays de clase `uint8`. Notar que los valores negativos no se redondean a cero como con la función `imsubtract`.

```
X = uint8([ 255 10 75; 44 225 100]);
Y = uint8([ 50 50 50; 50 50 50 ]);
Z = imabsdiff(X,Y)
S = imsubtract(X,Y)

Z =
    205     40     25
       6   175     50

S =
    205      0     25
       0   175     50
```

Ejemplo 2: Mostrar la diferencia en valor absoluto entre una imagen filtrada y la original.

```
I = imread('cameraman.tif');
J = uint8(filter2(fspecial('gaussian'), I));
K = imabsdiff(I,J);

% Mostrar la diferencia escalando los datos automáticamente con [].

imshow(K, [])
```



Figura f.179 - Imagen original (I)

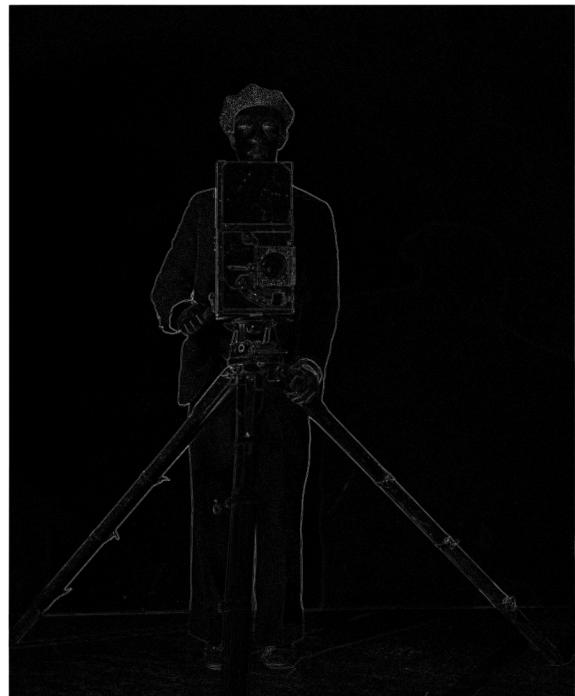


Figura f.231 - Imagen diferencia (K)

Funciones relacionadas

`imadd, imcomplement, imdivide, imlincomb, immultiply, imsubtract, ippl`

imadd

Juntar dos imágenes o sumar una constante a una imagen

Sintaxis

```
Z = imadd(X,Y)
```

Descripción

`Z = imadd(X,Y)` suma cada elemento del array `X` con su elemento correspondiente en el array `Y` devolviendo la suma en array `Z`. `X` e `Y` son arrays numéricos reales no dispersos de la misma clase y tamaño. `Y` también puede ser un escalar de clase `double`. `Z` tiene el mismo tamaño y clase que `X`, salvo cuando `X` es de clase `logical`, en cuyo caso `Z` es de clase `double`.

Si `X` e `Y` son arrays de enteros se truncarán los elementos a la salida que excedan el rango de los enteros y se redondearán los valores fraccionarios.

Ejemplos de uso

Ejemplo 1: Juntar dos arrays de clase `uint8`. Notar cómo se truncan los valores que exceden de 255.

```
X = uint8([ 255 0 75; 44 225 100; 12 35 44])
Y = uint8([ 50 50 50; 50 50 50; 50 50 50])
Z = imadd(X,Y)
```

```
X =
 255      0     75
  44    225    100
   12      35     44

Y =
   50      50     50
   50      50     50
   50      50     50

Z =
 255      50    125
   94    255    150
   62      85     94
```

Ejemplo 2: Juntar dos imágenes y especificar una clase de salida.

```
I = imread('cameraman.tif');
J = imread('puntos.tif');
K = imadd(I,J,'uint16');
imshow(K,[])
```



Figura f.179 - Imagen 1 (I)

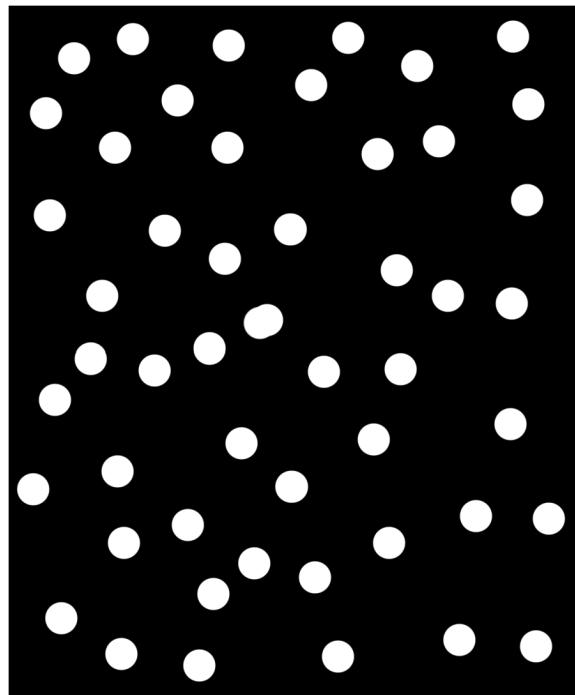


Figura f.232 - Imagen 2 (J)



Figura f.233 - Unión de imágenes (K)

Ejemplo 3: Sumar una constante a una imagen.

```
I = imread('puntos.tif');
J = imadd(I,50);
imshow(I)
figure, imshow(J)
```

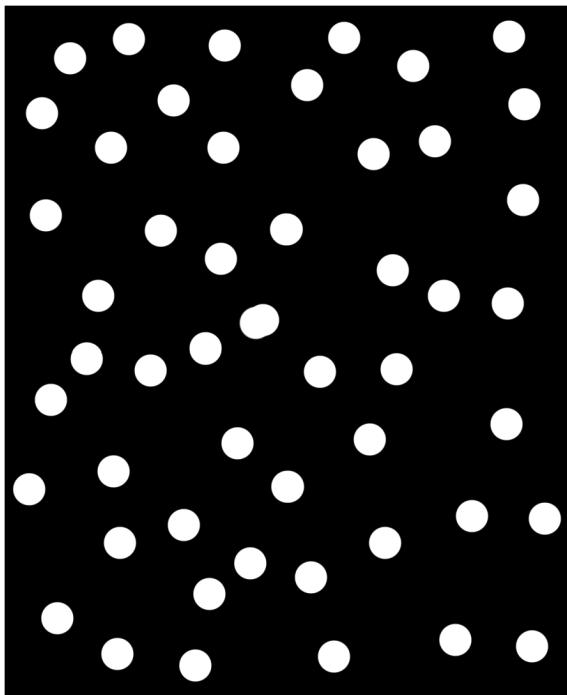


Figura f.232 - Imagen original (I)

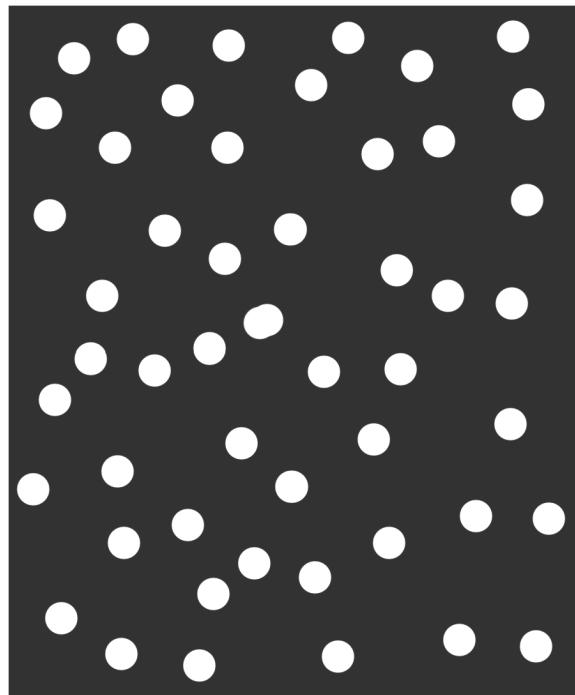


Figura f.234 - Imagen con añadido de 50 (J)

Funciones relacionadas

imabsdiff, imcomplement, imdivide, imlincomb, immultiply, imsubtract,
ippl

imadjust

Ajuste de los valores de intensidad de una imagen o de un mapa de color a un rango especificado

Introducción

La función `imadjust` suele utilizarse tanto para mostrar imágenes dentro de un rango de intensidad determinado como para la mejora simple del contraste de una imagen mediante la expansión de su histograma. Otras alternativas más complejas y específicas para la mejora del contraste son las funciones `histeq` y `adapthisteq`, que realizan una ecualización del histograma de la imagen y que resultan algo más automáticas. Más información y comparación en la página de la función `adapthisteq`.

Sintaxis

```
J = imadjust(I)
J = imadjust(I,[low_in; high_in],[low_out; high_out])
J = imadjust(I,[low_in; high_in],[low_out; high_out],gamma)
newmap = imadjust(map,[low_in; high_in],[low_out; high_out],gamma)
RGB2 = imadjust(RGB1,...)
```

Descripción

`J = imadjust(I)` ajusta los valores de intensidad de la imagen en escala de grises `I` saturando el 1% de los datos con mayor y menor intensidad (2% en total) usando los límites obtenidos automáticamente por la función `stretchlim` y guardando los valores en `J`. Esto incrementa el contraste de la imagen de salida `J`. Esta sintaxis es equivalente a hacer `imadjust(I,stretchlim(I))`.

`J = imadjust(I,[low_in; high_in],[low_out; high_out])` pasa los valores de intensidad de `I` a nuevos valores en `J`, saturando a `low_out` los valores menores que `low_in`, a `high_out` los valores por encima de `high_in` y convirtiendo linealmente los valores intermedios entre `low_in` y `high_in` a valores entre `low_out` y `high_out`. Los valores para `low_in`, `high_in`, `low_out`, y `high_out` deben estar en el rango `[0,1]` independientemente del tipo de imagen y de sus valores de intensidad. Estos límites se adaptan automáticamente al tipo de imagen, multiplicándose por 255 para imágenes de clase `uint8`, por 65535 para imágenes de clase `uint16`, etc. Se puede omitir el intervalo de

entrada o salida mediante una matriz vacía ([]), en cuyo caso se tomarán los valores de intervalo por defecto que son [0, 1].

`J = imadjust(I,[low_in; high_in],[low_out; high_out],gamma)` convierte los valores en `I` a nuevos valores en `J`, donde `gamma` especifica la forma de la curva que describe la relación entre los valores en `I` y `J`. Si `gamma` es menor que 1, la conversión se hace hacia valores de salida mayores (más brillantes) y si `gamma` es mayor que 1 hacia valores de salida menores (más oscuros). Si se omite el parámetro `gamma`, por defecto vale 1 (mapeado lineal).

`newmap = imadjust(map,[low_in; high_in],[low_out; high_out],gamma)` transforma el mapa de color de una imagen indexada especificado en `map`. Si `low_in`, `high_in`, `low_out`, `high_out` y `gamma` son escalares, entonces se realiza la misma conversión a las componentes roja, verde y azul. Se pueden usar vectores de 1-por-3 para definir mapeados diferentes en cada canal. El mapa reescalado `newmap` tendrá el mismo tamaño que el mapa original `map`.

`RGB2 = imadjust(RGB1,...)` realiza el ajuste en cada plano de la imagen (rojo, verde y azul) de la imagen RGB especificada en `RGB1`. Al igual que en el ajuste del mapa de color, al especificar las variables `low_in`, `high_in`, `low_out`, `high_out` y `gamma` se pueden definir escalares para especificar el mismo mapeado para todos los planos, o matrices de 1-por-3 para mapeados diferentes en cada plano.

Nota: Si `high_out < low_out`, la imagen de salida se revertirá, como en un negativo fotográfico.

La imagen de entrada puede ser de clase `uint8`, `uint16`, `int16`, `single`, o `double`. La imagen de salida será de la misma clase que la de entrada. Los mapas de color en la entrada y salida son de clase `double`.

Ejemplos de uso

Ejemplo 1: Comprobar el funcionamiento de la función `imadjust` ajustando un array de intensidades usando límites de entrada de valor inferior 0.4 (0.2 * 255 = 102) y superior 0.6 (0.6 * 255 = 153) y límites de salida por defecto ([0 1])

```
I = reshape(uint8(linspace(1,255,50)),[5 10])
K = imadjust(I,[0.4 0.6],[])
```

```
I =
  1   27   53   79   105   131   157   182   208   234
  6   32   58   84   110   136   162   188   214   239
 11   37   63   89   115   141   167   193   219   245
 17   42   68   94   120   146   172   198   224   250
 22   48   74   99   125   151   177   203   229   255

K =
  0   0   0   0   15   145   255   255   255   255
  0   0   0   0   40   170   255   255   255   255
  0   0   0   0   65   195   255   255   255   255
  0   0   0   0   90   220   255   255   255   255
  0   0   0   0  115   245   255   255   255   255
```

Ejemplo 2: Ajustar una imagen en escala de grises de bajo contraste.

```
I = imread('espacio.tif');
imshow(I)
J = imadjust(I);
figure, imshow(J)
```

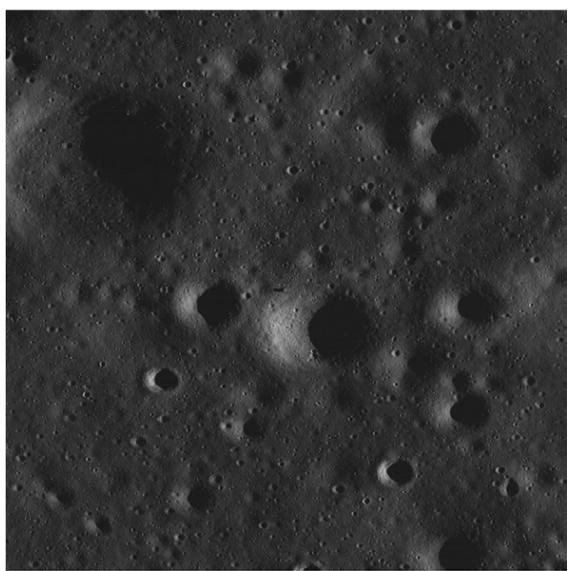


Figura f.1 - Imagen original (I) sin procesar

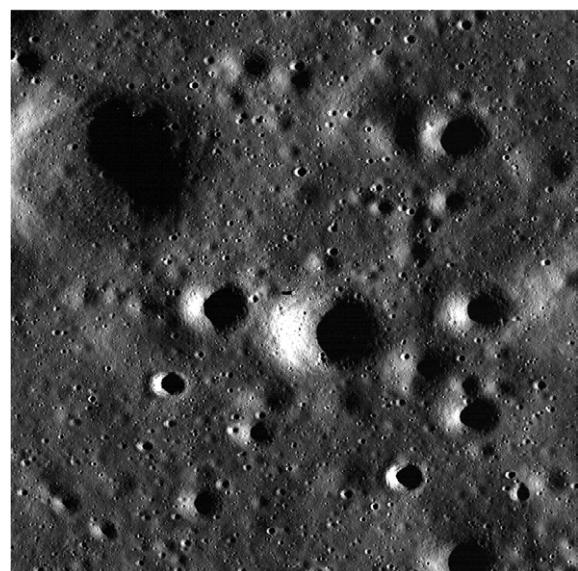


Figura f.5 - Imagen ajustada (J) con imadjust

Ejemplo 3: Ajustar una imagen RGB usando los límites de entrada y salida.

```
RGB1 = imread('Nurnberg.tif');
imshow(RGB1)
RGB2 = imadjust(RGB1,[.2 .2 0; .7 .8 1],[]);
figure, imshow(RGB2)
```



Figura f.1 - Imagen original (RGB1) sin procesar



Figura f.235 - Imagen ajustada (RGB2) con imadjust

Funciones relacionadas

adaphisteq, brighten, histeq, stretchlim

ImageAdapter

Interfaz para la lectura y escritura de archivos por bloques

Sintaxis

```
adapter = ClassName( . . . )
```

Descripción

`ImageAdapter` es una clase abstracta que especifica la interfaz para la lectura y escritura de archivos por bloques o regiones, definiendo los métodos que la función `blockproc` utilizará para leer y escribir imágenes en el disco. Aunque la función `blockproc` solo admite determinados tipos de archivos, se puede conseguir trabajar con otro tipo de imágenes construyendo para ellos una clase que herede de la clase `ImageAdapter` y encapsule el código necesario.

Para poder escribir una clase `Image Adapter` para un formato de archivo determinado se debe poder:

- Consultar el tamaño del archivo en disco.
- Leer un bloque rectangular del archivo.

Al heredar de la superclase `ImageAdapter` la nueva clase podrá:

- Interactuar con la función `blockproc`.
- Definir propiedades comunes de las clases `ImageAdapter`.
- Definir la interfaz que `blockproc` utilizará para leer y escribir en otros archivos.

Por ejemplo, al crear una nueva clase `ImageAdapter` llamada `ClassName` se tendría que programar un archivo `ClassName.m` con el código correspondiente, en el que se definiría la clase (su nombre y que hereda de la superclase `ImageAdapter`), sus propiedades y sus métodos (constructor de clase, lectura de regiones, cierre de clase...). Como ejemplo se puede consultar la clase `LanAdapter` en MATLAB, creada para los archivos en formato LAN mediante la sintaxis `open LanAdapter.m`. Para construir un objeto de la clase `LanAdapter` que se pueda utilizar como fuente de entrada a la función `blockproc` se debe

utilizar el constructor de la clase, utilizando la sintaxis ADAPTER = LANADAPTER(archivo).

De forma más general, `adapter = ClassName(...)` construye un objeto de la clase `ClassName` y maneja la inicialización del objeto, la apertura o creación del archivo y especifica los valores iniciales de las propiedades de la clase. El constructor de la clase acepta cualquier número de argumentos. Las propiedades heredadas de la superclase `ImageAdapter` y que se pueden definir al construir cualquier objeto son:

Colormap	Especifica un mapa de color. Utilizar esta propiedad al trabajar con imágenes indexadas. El valor de la propiedad debe ser una matriz de M-por-3 de números reales. (Por defecto: Valor vacío <code>[]</code> que indica o una imagen en escala de grises, una imagen de clase <code>logical</code> o una imagen de color verdadero)
ImageSize	Guarda el tamaño de la imagen. Cuando se construye una nueva clase que hereda de <code>ImageAdapter</code> , se debe definir la propiedad <code>ImageSize</code> en el constructor de la clase. El valor especificado debe ser un vector de 2 o 3 elementos del tipo <code>[rows cols]</code> o <code>[rows cols bands]</code> , donde <code>rows</code> indica el alto y <code>cols</code> indica el ancho.

Las clases `ImageAdapter` (para cada tipo de archivo) que heredan de la superclase `ImageAdapter` deben implementar obligatoriamente los métodos `readRegion` (lectura de una región de la imagen) y `close` (cerrar objeto `ImageAdapter`) para permitir una lectura básica de imágenes basada en regiones. El método `writeRegion` permite la escritura incremental de imágenes basándose en regiones y es opcional. Las clases `ImageAdapter` que no implementan el método `writeRegion` son clases de solo lectura.

Funciones relacionadas

[blockproc](#)

imageinfo

Herramienta de información de imagen

Sintaxis

```
imageinfo
imageinfo(h)
imageinfo(filename)
imageinfo(info)
imageinfo(himage,filename)
imageinfo(himage,info)
hfig = imageinfo(...)
```

Descripción

`imageinfo` crea una herramienta de información de imagen asociada con la imagen de la figura actual. La herramienta muestra información de los atributos básicos de la imagen en una figura separada. `imageinfo` obtiene la información consultando el objeto de imagen `CData`.

La siguiente tabla muestra la información básica incluida al utilizar la herramienta. Notar que la herramienta contiene cuatro o seis campos en función del tipo de imagen.

Atributo	Descripción
Width	Número de columnas de la imagen.
Height	Número de filas en la imagen.
Class	Tipo de dato usado en la imagen, como p. ej. <code>uint8</code> . Nota: Para imágenes de clase <code>single</code> o <code>int16</code> , <code>imageinfo</code> devuelve un valor de clase de <code>double</code> porque los objetos de imagen convierten el <code>CData</code> de estas imágenes a clase <code>double</code> .
Image type	Uno de los tipos de imagen identificados: ' <code>intensity</code> ' si es una imagen de intensidades, ' <code>truecolor</code> ' si es una imagen en color verdadero, ' <code>binary</code> ' si es una imagen binaria, o ' <code>indexed</code> ' si es una imagen indexada.
Minimum intensity or index	Para imágenes en escala de grises este valor representa la menor intensidad de pixel encontrada. Para imágenes indexadas el valor representa el menor valor de índice en un mapa de color. Atributo no incluido en imágenes binarias o de color verdadero.
Maximum	Para imágenes en escala de grises este valor representa la mayor

intensity or index	<p>intensidad de pixel encontrada.</p> <p>Para imágenes indexadas el valor representa el mayor valor de índice en un mapa de color.</p> <p>Atributo no incluido en imágenes binarias o de color verdadero.</p>
--------------------	--

`imageinfo(h)` crea una herramienta de información de imagen asociada con `h`, donde `h` es un puntero a una figura, eje u objeto de imagen.

`imageinfo(filename)` crea una herramienta de información de imagen que lee los metadatos del archivo gráfico `filename`. No hace falta mostrar la imagen en una ventana. `filename` puede ser de cualquier tipo registrado en el registro de formatos de archivo `imformats` por lo que su información puede leerse con `imfinfo`. `filename` puede ser también de tipo DICOM, NITF, Interfile, o Analyze.

`imageinfo(info)` crea una herramienta de información de imagen leyendo los metadatos de la imagen de la estructura `info`, una estructura devuelta por las funciones `imfinfo`, `dicominfo`, `nitfinfo` `interfileinfo`, o `analyze75info`. `info` puede ser también una estructura creada por el usuario.

`imageinfo(himage,filename)` crea una herramienta de información de imagen que lee la información de los atributos básicos de la imagen desde el puntero `himage` al objeto gráfico y lee los metadatos de la imagen del archivo gráfico `filename`.

`imageinfo(himage,info)` crea una herramienta de información de imagen que lee la información de los atributos básicos de la imagen desde el puntero `himage` al objeto gráfico y lee los metadatos de la imagen de la estructura `info`.

`hfig = imageinfo(...)` devuelve un puntero a la figura de la herramienta de información de imagen.

Ejemplos de uso

Ejemplo 1: Mostrar la información básica de una imagen recién mostrada.

```
imshow('cameraman.tif')
imageinfo;
```



Figura f.179 - Imagen (cameraman.tif)

Image details (Figure 1)	
Attribute	Value
Width (columns)	886
Height (rows)	1084
Class	uint8
Image type	intensity
Minimum intensity	0
Maximum intensity	255

Figura f.236 - Información básica de la imagen

Ejemplo 2: Mostrar la información completa de una imagen y además mostrar sus metadatos.

```
h = imshow('cameraman.tif')
info = imfinfo('cameraman.tif');
imageinfo(h,info);
```

Funciones relacionadas

`analyze75info`, `dicominfo`, `imattributes`, `imfinfo`, `imformats`, `imtool`,
`interfileinfo`, `nitfinfo`

imagemodel

Creación de un objeto modelo de imagen

Introducción

Un objeto modelo de imagen guarda cierta información de una imagen: clase, tipo, rango de intensidades, anchura, altura y mínimo y máximo valor de intensidad. El objeto acepta métodos que se pueden utilizar para acceder y mostrar la información. Para obtener la lista de métodos se debe escribir en MATLAB el código `methods imagemodel` o también se puede escribir `help imagemodel/método` para obtener más información a cerca de un método determinado. Estos son los métodos aceptados y los valores devueltos:

`str = getClassType(imgmodel)`

Clase de la imagen.

`disp_range = getDisplayRange(imgmodel)`

Rango de la imagen, si esta es de intensidades.

`height = getImageHeight(imgmodel)`

Número de filas.

`str = getImageType(imgmodel)`

Tipo de imagen.

`width = getImageWidth(imgmodel)`

Número de columnas.

`minval = getMinIntensity(imgmodel)`

Mínimo valor de intensidad de la imagen.

`maxval = getMaxIntensity(imgmodel)`

Máximo valor de intensidad de la imagen.

`fun = getNumberFormatFcn(imgmodel)`

Puntero a una función que convierte un valor numérico en una cadena de texto. Por ejemplo, `str = fun(getPixelValor(imgmodel, 100, 100))` convierte el valor numérico devuelto por la función `getPixelValor` a una cadena de texto.

`str = getPixelInfoString(imgmodel, row, column)`

Cadena de texto del valor del pixel especificado.

`fun = getPixelRegionFormatFcn(imgmodel)`

Puntero a una función que da formato al valor del pixel especificado.

`val = getPixelValor(imgmodel, row, column)`

Array numérico del valor del pixel especificado.

```
str = getDefectoPixelInfoString(imgmodel)
```

Cadena de texto que indica el tipo de pixel.

```
str = getDefectPixelRegionString(imgmodel)
```

Cadena de texto que indica el tipo de información mostrada en la región seleccionada.

```
val = getScreenPixelRGBValor(imgmodel, row, col)
```

Valor en pantalla del pixel especificado.

Sintaxis

```
imgmodel = imagemode(himage)
```

Descripción

`imgmodel = imagemode(himage)` crea un objeto modelo de imagen asociado con una imagen destino. `himage` es un puntero a un objeto de imagen o un array de punteros a objetos de imagen.

`imagemode` devuelve un objeto modelo de imagen o un array de objetos modelo de imagen si `himage` es un array de objetos de imagen.

`imagemode` trabaja consultando la propiedad de objeto de imagen `CData`. Para imágenes tipo `single` o `int16` el objeto de imagen convierte su `CData` a clase `double`.

Ejemplos de uso

Ejemplo 1: Crear un objeto modelo de imagen de una imagen.

```
h = imshow('peppers.png')
im = imagemode(h);
```

Funciones relacionadas

```
getimagemode
```

imattributes

Atributos de una imagen

Sintaxis

```
atrs = imattributes
atrs = imattributes(himage)
atrs = imattributes(imgmodel)
```

Descripción

`atrs = imattributes` devuelve información sobre la imagen de la figura actual consultando la propiedad `CData` del objeto de imagen. Si la figura actual no contiene ninguna imagen, la función devuelve un array vacío.

`atrs = imattributes(himage)` devuelve información sobre la imagen especificada por `himage`, un puntero a un objeto de imagen.

`imattributes` devuelve la información de los atributos de la imagen en la variable `atrs`, un array de celdas de tamaño 4-por-2 o 6-por-2 dependiendo del tipo de imagen. La primera columna del array de celdas contiene el nombre del atributo guardado como cadena de texto. La segunda columna contiene el valor del atributo también como cadena de texto. La siguiente tabla muestra los atributos en el orden en que aparecen en el array de celdas:

Nombre del atributo	Valor
Width	Número de columnas de la imagen.
Height	Número de filas en la imagen.
Class	Tipo de dato usado en la imagen, como p.ej. <code>uint8</code> . Nota: Para imágenes de clase <code>single</code> o <code>int16</code> , <code>imageinfo</code> devuelve un valor de clase de <code>double</code> porque los objetos de imagen convierten el <code>CData</code> de estas imágenes a clase <code>double</code> .
Image type	Uno de los tipos de imagen identificados: ' <code>intensity</code> ' si es una imagen de intensidades, ' <code>truecolor</code> ' si es una imagen en color verdadero, ' <code>binary</code> ' si es una imagen binaria, o ' <code>indexed</code> ' si es una imagen indexada.
Minimum	Para imágenes en escala de grises este valor representa la menor

intensity or index	intensidad de pixel encontrada. Para imágenes indexadas el valor representa el menor valor de índice en un mapa de color. Atributo no incluido en imágenes binarias o de color verdadero.
Maximum intensity or index	Para imágenes en escala de grises este valor representa la mayor intensidad de pixel encontrada. Para imágenes indexadas el valor representa el mayor valor de índice en un mapa de color. Atributo no incluido en imágenes binarias o de color verdadero.

`attrs = imattributes(imgmodel)` devuelve información a cerca de la imagen representada por el objeto modelo de imagen `imgmodel`.

Ejemplos de uso

Ejemplo 1: Leer los atributos de una imagen en escala de grises.

```
h = imshow('liftingbody.png')
attrs = imattributes(h)

attrs =
    'Width (columns)'      '512'
    'Height (rows)'        '512'
    'Class'                'uint8'
    'Image type'           'intensity'
    'Minimum intensity'   '0'
    'Maximum intensity'   '255'
```

Funciones relacionadas

`imgmodel`

imbothat

Filtrado tipo bottom-hat

Introducción

En morfología matemática y procesado digital de imágenes la transformada *bottom-hat* o *black top-hat* (sombrero de copa negro), dual a la transformada *white top-hat* o *top-hat* (sombrero de copa blanco), se define como la diferencia entre la clausura de una imagen por un elemento estructurante (la colección de partes del fondo de la imagen que corresponden con un elemento estructurante o patrón particular) y la imagen original. Es una dilatación seguida de una erosión.

Las transformadas *top-hat* se utilizan en numerosas tareas dentro del procesado digital de imágenes como por ejemplo la extracción de características, ecualización del fondo o mejora de la imagen. En concreto, la transformada *bottom-hat* devuelve una imagen con los objetos que son menores que el elemento estructurante y que son más oscuros que sus vecinos (se debe aumentar el tamaño del elemento estructurante para extraer objetos mayores). Es por tanto útil para extraer pequeños objetos o detalles oscuros sobre fondo claro y también para corregir la iluminación desigual cuando el fondo es claro. Además, se puede mejorar el contraste de una imagen sumando la imagen original y la imagen filtrada con *top-hat* y posteriormente restándole la imagen filtrada con *bottom-hat*. [22] [23]

Sintaxis

```
IM2 = imbothat(IM,SE)
IM2 = imbothat(IM,NHOOD)
```

Descripción

`IM2 = imbothat(IM,SE)` realiza un filtrado tipo *bottom-hat* a la imagen binaria o en escala de grises `IM` devolviendo la imagen filtrada `IM2`. El argumento `SE` es un elemento estructurante como el devuelto por la función `strel`. `SE` debe ser un elemento estructurante individual, no un array de elementos.

`IM2 = imbothat(IM,NHOOD)` realiza un filtrado tipo *bottom-hat* donde `NHOOD` es un array de 0's y 1's que especifica el tamaño y forma del elemento estructurante. Esto equivale a `imbothat(IM,strel(NHOOD))`.

`IM` puede ser de clase `numeric` o `logical` y debe ser no disperso. La imagen de salida será del mismo tipo que la de entrada. Si la entrada es binaria (`logical`) el elemento estructurante debe ser plano.

Ejemplos de uso

Ejemplo 1: Separar los objetos del fondo con filtrado *bottom-hat*, variando el tamaño del elemento estructurante en forma de disco. Observar cómo el fondo se hace uniforme.

```
I = imread('cacao.tif'); imshow(I)
se = strel('disk',3);
B1 = imbothat(I,se); figure, imshow(B1)
se = strel('disk',10);
B2 = imbothat(I,se); figure, imshow(B2)
se = strel('disk',30);
B3 = imbothat(I,se); figure, imshow(B3)
```



Figura f.237 - Imagen original (I) sin procesar



Figura f.238 - Imagen procesada (B1) con imbothat y elemento estructurante disco de 3



Figura f.239 - Imagen procesada (B2) con imbothat y elemento estructurante disco de 10



Figura f.240 - Imagen procesada (B3) con imbothat y elemento estructurante disco de 30

Ejemplo 2: Quedarse solo con los círculos más pequeños de la imagen.

```
I = imread('puntos.tif'); imshow(I)
se = strel('disk',14);
B = imbothat(I,se); figure, imshow(B)
```



Figura f.241 - Imagen original (I) sin procesar



Figura f.242 - Imagen procesada (B) con imbothat

Ejemplo 3: Aumentar el contraste de una imagen utilizando filtrado *top-hat* y *bottom-hat*.

```
I = imread('espacio.tif'); imshow(I)
se = strel('disk',3);
T = imtophat(I,se);
B = imbothat(I,se);

% Añadimos la imagen original I a la imagen filtrada top-hat y después le
restamos la imagen filtrada bottom-hat para aumentarle el contraste.

J = imsubtract(imadd(I,T), B); figure, imshow(J)
```



Figura f.1 - Imagen original (I) sin procesar

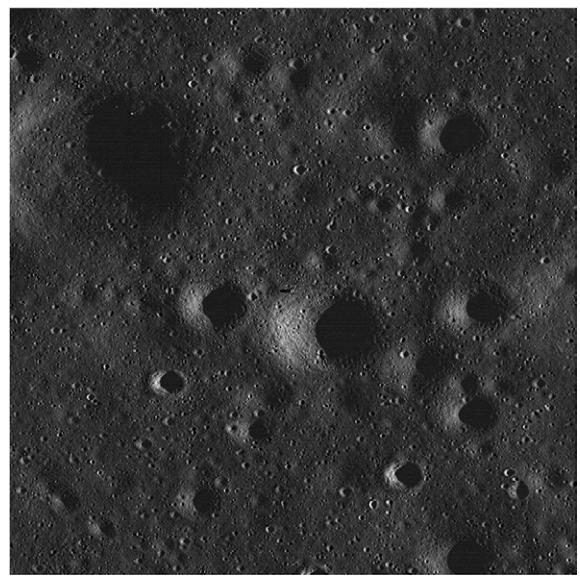


Figura f.243 - Imagen procesada (J) aumentándole el contraste con imtophat e imbothat

Funciones relacionadas

imtophat, strel

imclearborder

Eliminación de estructuras suaves conectadas al borde de una imagen

Sintaxis

```
IM2 = imclearborder(IM)
IM2 = imclearborder(IM,conn)
```

Descripción

`IM2 = imclearborder(IM)` elimina estructuras que son más suaves que sus vecinos y que están conectadas al borde de la imagen (útil para limpiar el borde de la imagen). `IM` puede ser una imagen binaria o en escala de grises. La imagen de salida `IM2` será igualmente binaria o en escala de grises. La conectividad por defecto es 8 para dos dimensiones, 26 para tres dimensiones y `conndef(ndims(BW), 'maximal')` para mayores dimensiones.

Nota: Para imágenes en escala de grises, `imclearborder` tiende a reducir el nivel de intensidad general además de suprimir estructuras de borde.

`IM2 = imclearborder(IM,conn)` especifica la conectividad deseada. `conn` puede tener cualquiera de los siguientes valores escalares (siempre simétrica respecto su elemento central):

Valor	Significado
Conejividades Bidimensionales	
4	Vecindad de 4 píxeles: Un par de píxeles colindantes forman parte del mismo objeto solo si están activos y conectados en vertical u horizontal.
8	Vecindad de 8 píxeles: Un par de píxeles colindantes forman parte del mismo objeto solo si están activos. (Por defecto)
Conejividades Tridimensionales	
6	Vecindad de 6 píxeles.
18	Vecindad de 18 píxeles.
26	Vecindad de 26 píxeles. (Por defecto)

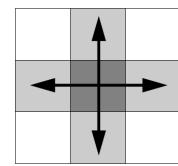


Figura f.32 - vecindad 4

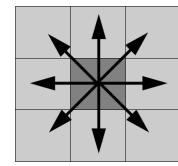


Figura f.33 - vecindad 8

Nota: La conectividad se puede definir para cualquier dimensión de forma más general utilizando para `conn` una matriz de 3-por-3-por...-por-3 de ceros y unos. Los unos definen las localizaciones de los píxeles vecinos relativas al elemento central de `conn`. Por ejemplo, `conn = ones(3)` define una conectividad de 8 píxeles.

Puede que un pixel en el borde de una imagen no se considere un pixel del borde si se elige una conectividad no por defecto. Por ejemplo, si `conn = [0 0 0; 1 1 1; 0 0 0]`, los elementos en la primera y última fila no se considerarán píxeles de borde porque según la conectividad, no están conectados a la región de fuera de la imagen.

`IM` puede ser un array de tipo `numeric` o `logical` de cualquier dimensión y debe ser no disperso y real. `IM2` será de la misma clase que `IM`.

Ejemplos de uso

Ejemplo 1: Utilizar `imclearborder` con diferentes conectividades y analizar los resultados partiendo del siguiente array.

```
BW =
    0     0     0     0     0     0     0     0     0
    0     0     0     0     0     0     0     0     0
    0     0     0     0     0     0     0     0     0
    1     0     0     1     1     1     0     0     0
    0     1     0     1     1     1     0     0     0
    0     0     0     1     1     1     0     0     0
    0     0     0     0     0     0     0     0     0
    0     0     0     0     0     0     0     0     0
    0     0     0     0     0     0     0     0     0
```

Usando una conectividad de 4 elementos, el pixel en la posición $(5, 2)$ no se considera conectado al pixel del borde $(4, 1)$, así que no se elimina.

```
BWcl = imclearborder(BW, 4)
```

```
BWcl =
    0     0     0     0     0     0     0     0     0
    0     0     0     0     0     0     0     0     0
    0     0     0     0     0     0     0     0     0
    0     0     0     1     1     1     0     0     0
    0     1     0     1     1     1     0     0     0
    0     0     0     1     1     1     0     0     0
    0     0     0     0     0     0     0     0     0
    0     0     0     0     0     0     0     0     0
    0     0     0     0     0     0     0     0     0
```

Usando una conectividad de 8 elementos, el pixel en la posición $(5, 2)$ se considera conectado al pixel del borde $(4, 1)$, así que ambos se eliminan.

```
BWc2 = imclearborder(BW, 8)
```

```
BWc2 =
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 0 0 0
0 0 0 1 1 1 0 0 0
0 0 0 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
```

Funciones relacionadas

[conndef](#)

imclose

Cierre morfológico o clausura de una imagen

Sintaxis

```
IM2 = imclose(IM,SE)
IM2 = imclose(IM,NHOOD)
```

Descripción

`IM2 = imclose(IM,SE)` cierra morfológicamente la imagen binaria o en escala de grises `IM`, devolviendo la imagen procesada `IM2`. El elemento estructurante `SE` debe ser individual, no un array de objetos. La operación morfológica de cierre o clausura de una imagen consiste una dilatación seguida por una erosión usando el mismo elemento estructurante en ambas operaciones.

`IM2 = imclose(IM,NHOOD)` cierra la imagen utilizando el elemento estructurante `strel(NHOOD)`, donde `NHOOD` es un array de 0's y 1's que especifican la vecindad o patrón del elemento estructurante.

`IM` puede ser de cualquier clase `numeric` o `logical` y de cualquier dimensión y debe ser no disperso. Si `IM` es `logical`, entonces `SE` debe ser plano. `IM2` será de la misma clase que `IM`.

Ejemplos de uso

Ejemplo 1: Utilizar `imclose` para unir los círculos de la imagen rellenando el hueco entre ellos y suavizando los bordes exteriores.

```
I = imread('circulos.tif');
imshow(I)

% Crear un elemento estructurante en forma de disco para preservar la
naturaleza circular del objeto. Usamos un radio de 10 píxeles para
conseguir llenar los huecos, que si fueran más grandes, habría que
aumentarlo.

se = strel('disk',10);

% Cerrar la imagen morfológicamente.

closeI = imclose(I,se);
figure, imshow(closeI)
```

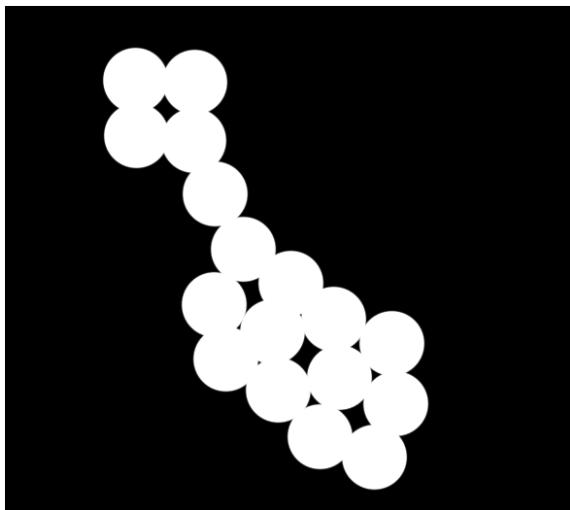


Figura f.244 - Imagen original (*I*) sin procesar

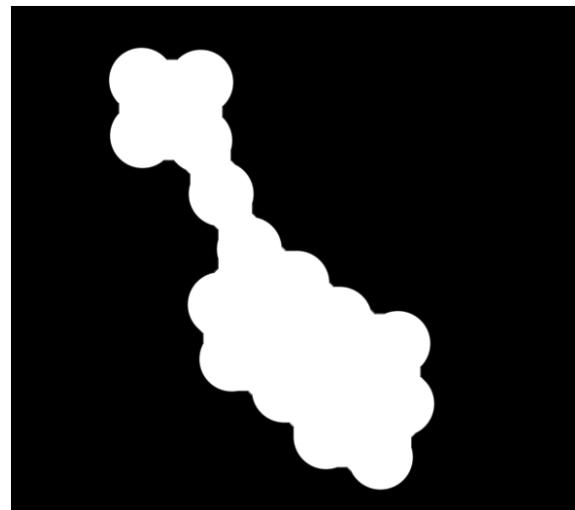


Figura f.245 - Imagen procesada (`closeI`)

Funciones relacionadas

`imdilate`, `imerode`, `imopen`, `strel`

imcolormaptool

Herramienta de selección de mapa de color

Sintaxis

```
imcolormaptool  
imcolormaptool(hclientfig)  
hfig = imcolormaptool(...)
```

Descripción

La herramienta de selección del mapa de color es una herramienta interactiva de selección del mapa de color que permite cambiar el mapa de color de una figura (la actual) seleccionando el mapa de color de una lista de funciones o de variables en el espacio de trabajo o introduciendo una expresión específica.

`imcolormaptool` abre la herramienta de selección del mapa de color en una nueva figura que estará asociada con la figura destino.

`imcolormaptool(hclientfig)` abre la herramienta de selección del mapa de color utilizando `hclientfig` como figura destino. `hclientfig` debe contener o una imagen en escala de grises o una indexada.

`hfig = imcolormaptool(...)` devuelve un puntero a la herramienta de selección del mapa de color en la variable `hfig`.

Ejemplos de uso

Ejemplo 1: Abrir la herramienta de selección de mapa de color después de abrir una imagen.

```
h = figure;  
imshow('cameraman.tif')  
imcolormaptool(h);
```



Figura f.179 - Imagen (cameraman.tif)

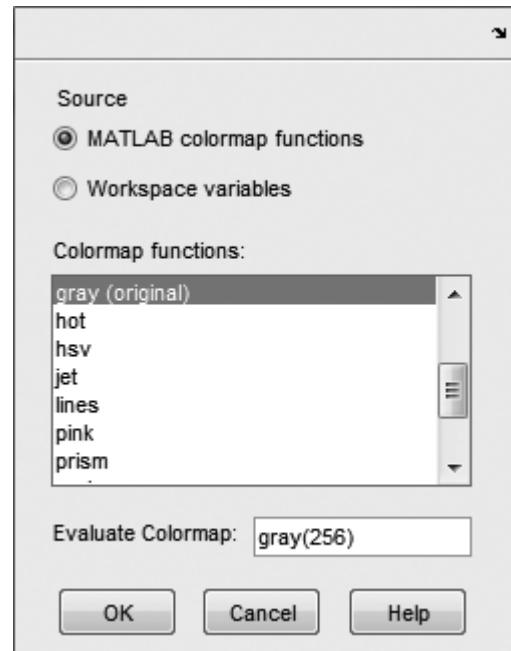


Figura f.246 - Herramienta de selección de mapa de color

Funciones relacionadas

colormap, imshow, imtool

imcomplement

Imagen complementaria

Sintaxis

```
IM2 = imcomplement(IM)
```

Descripción

`IM2 = imcomplement(IM)` calcula la imagen complementaria de la imagen `IM`. `IM` puede ser una imagen binaria, en escala de grises, o RGB. `IM2` será de la misma clase y tamaño que `IM`.

En la imagen complementaria de una imagen binaria, los ceros se convierten en unos y los unos en ceros; los colores blanco y negro se invierten. En la imagen complementaria de una imagen en escala de grises o RGB, cada valor de pixel se resta del máximo valor de pixel posible debido a la clase de la imagen y la diferencia se usa como valor de pixel en la imagen de salida. En este caso, las áreas oscuras se convierten en áreas más claras en la imagen de salida y las áreas más claras se convierten en más oscuras.

Si `IM` es una imagen en escala de grises o RGB de clase `double`, se puede usar la expresión `1 - IM` en vez de esta función. Si `IM` es una imagen binaria, se puede usar la expresión `~IM` en vez de esta función.

Ejemplos de uso

Ejemplo 1: Crear el complemento de un array de clase `uint8`.

```
X = uint8([ 255 10 75; 44 225 100]);
C = imcomplement(X)

C =
    0    245    180
   211     30    155
```

Ejemplo 2: Crear la imagen complementaria de una imagen binaria (invirtiendo el negro y el blanco).

```
I = imread('circulos.tif');
C = imcomplement(I);
imshow(C)
```

```
figure, imshow(C)
```

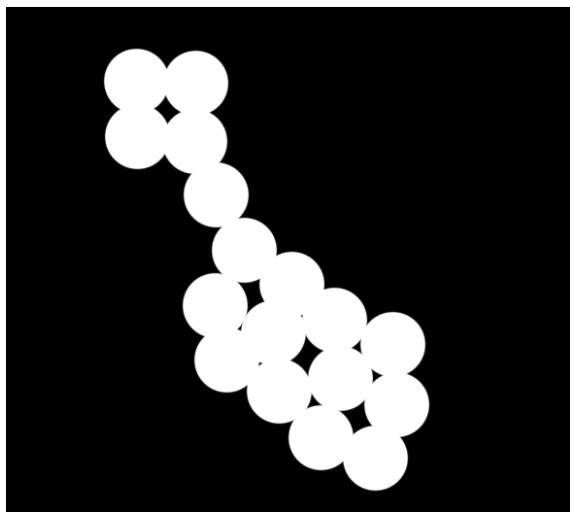


Figura f.245 - Imagen original (I) sin procesar

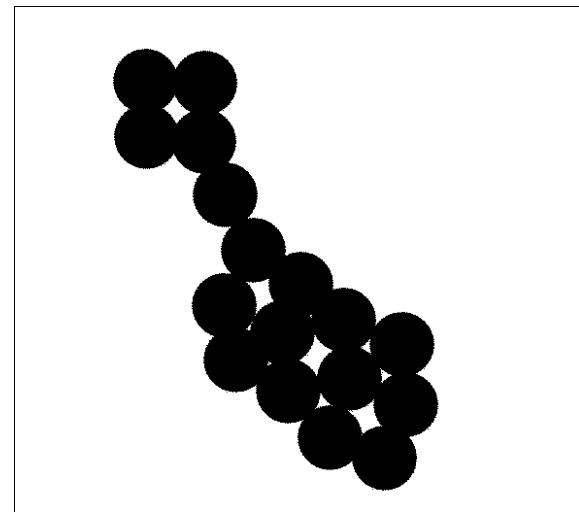


Figura f.247 - Imagen procesada (C) con imcomplement

Ejemplo 3: Crear la imagen complementaria de una imagen en escala de grises.

```
I = imread('cameraman.tif');  
C = imcomplement(I);  
imshow(I)  
figure, imshow(C)
```



Figura f.179 - Imagen original (I) sin procesar



Figura f.248 - Imagen procesada (C) con imcomplement

Funciones relacionadas

imabsdiff, imadd, imdivide, imlincomb, immultiply, imsubtract

imcontour

Dibujo del contorno de una imagen

Sintaxis

```
imcontour(I)
imcontour(I,n)
imcontour(I,v)
imcontour(x,y,...)
imcontour(...,LineSpec)
[C,handle] = imcontour(...)
```

Descripción

`imcontour(I)` crea una nueva figura dibujando el contorno de la imagen en escala de grises `I` y definiendo automáticamente los ejes para que su orientación y proporción coincidan con la imagen.

`imcontour(I,n)` crea una nueva figura dibujando el contorno de la imagen en escala de grises `I` y definiendo automáticamente los ejes para que su orientación y proporción coincidan con la imagen. `n` es el número de niveles de contorno equiespaciados en la figura; si se omite el argumento, el número de niveles y de valores de los niveles se eligen automáticamente.

`imcontour(I,v)` dibuja el contorno de `I` con líneas de contorno especificadas en el vector `v`. El número de niveles de contorno es igual a `length(v)`.

`imcontour(x,y,...)` usa los vectores `x` e `y` para especificar los límites en el eje `x` e `y` respectivamente.

`imcontour(...,LineSpec)` dibuja los contornos utilizando el tipo de línea y color especificados en la variable `LineSpec`. Los símbolos marcadores se ignoran.

`[C,handle] = imcontour(...)` devuelve la matriz de contorno `C` y un puntero a un objeto `hggroup`.

La imagen de entrada puede ser de clase `uint8`, `uint16`, `int16`, `single`, `double` o `logical`.

Ejemplos de uso

Ejemplo 1: Dibujar el contorno de una imagen.

```
I = imread('circuito.tif');
imshow(I)
figure, imcontour(I,3)
```

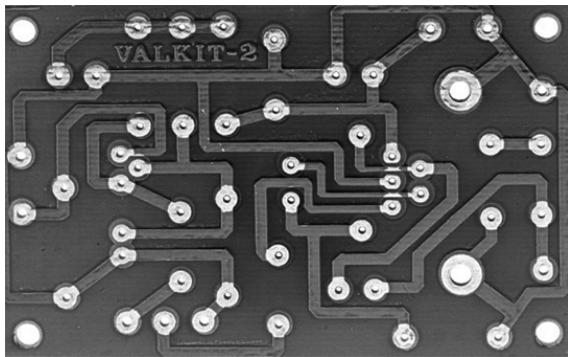


Figura f.249 - Imagen original (I) sin procesar

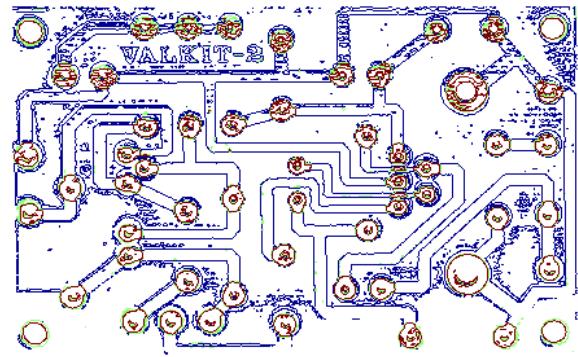


Figura f.250 - Contorno de la imagen con imcontour

Funciones relacionadas

clabel, contour, LineSpec

imcontrast

Herramienta de ajuste del contraste

Sintaxis

```
imcontrast  
imcontrast(h)  
hfigure = imcontrast(...)
```

Descripción

`imcontrast` crea una herramienta para el ajuste interactivo del contraste en una figura separada asociada con la imagen en escala de grises de la figura actual. Cuando se usa la herramienta, `imcontrast` ajusta el contraste de la imagen mostrada modificando la propiedad `CLim`. Para modificar los valores de los píxeles de la imagen actual se debe hacer clic en el botón **Adjust Data** (“Ajustar datos”). El botón no está disponible hasta que se haga un cambio al contraste de la imagen.

Nota: La herramienta de ajuste del contraste puede manejar imágenes en escala de grises de clase `double` y `single` con rangos de datos más allá que el rango por defecto, que en este caso es `[0,1]`. `imcontrast` fija los límites del histograma rellenando las fronteras superior e inferior para que cubra el rango de datos de la imagen.

`imcontrast(h)` crea la herramienta de ajuste de contraste asociada con la imagen especificada por el puntero `h`. `h` puede ser un puntero a una figura, eje u objeto de imagen. Si `h` es un eje o puntero a una figura, `imcontrast` usa la primera imagen devuelta por `findobj(H, 'Type', 'image')`.

`hfigure = imcontrast(...)` devuelve un puntero a la herramienta de ajuste del contraste.

Ejemplos de uso

Ejemplo 1: Abrir la herramienta de ajuste del contraste después de abrir una imagen.

```
I = imread('circuito.tif');  
imshow(I)  
imcontrast
```

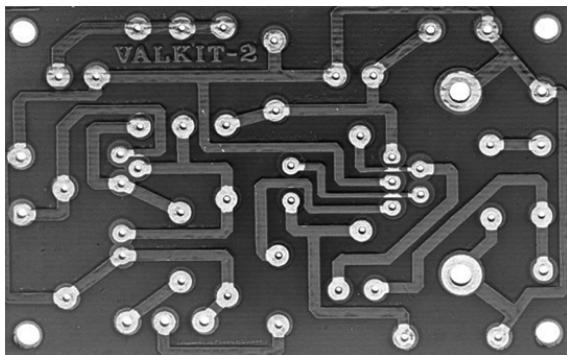


Figura f.249 - Imagen original (I) sin procesar

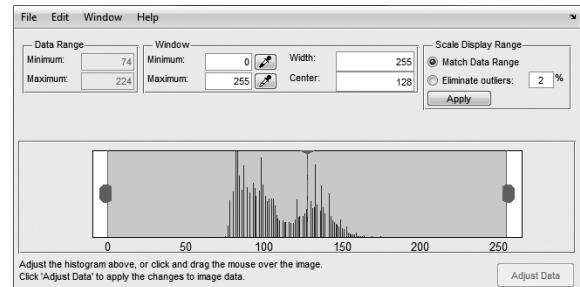


Figura f.251 - Herramienta de ajuste del contraste

Funciones relacionadas

`imadjust`, `imtool`, `stretchlim`

imcrop

Herramienta para el recorte de una imagen

Sintaxis

```
I = imcrop  
I2 = imcrop(I)  
X2 = imcrop(X, map)  
I = imcrop(h)  
I2 = imcrop(I, rect)  
X2 = imcrop(X, map, rect)  
[...] = imcrop(x, y,...)  
[I2 rect] = imcrop(...)  
[x,y,I2,rect] = imcrop(...)
```

Descripción

`I = imcrop` crea una herramienta interactiva para el recorte de una imagen asociada con la imagen de la figura actual. La herramienta consiste en un rectángulo de selección que es posible mover y redimensionar con el ratón para después recortar esa zona. Cuando la herramienta está activa, el puntero del ratón se convierte en una cruz al moverlo por la imagen. Al finalizar la selección mediante el ratón, se debe hacer doble clic para recortar la imagen por la selección o hacer clic en “**Crop Image**” (Recortar imagen) del menú contextual. `imcrop` devuelve la imagen recortada. La siguiente figura muestra la herramienta activa:

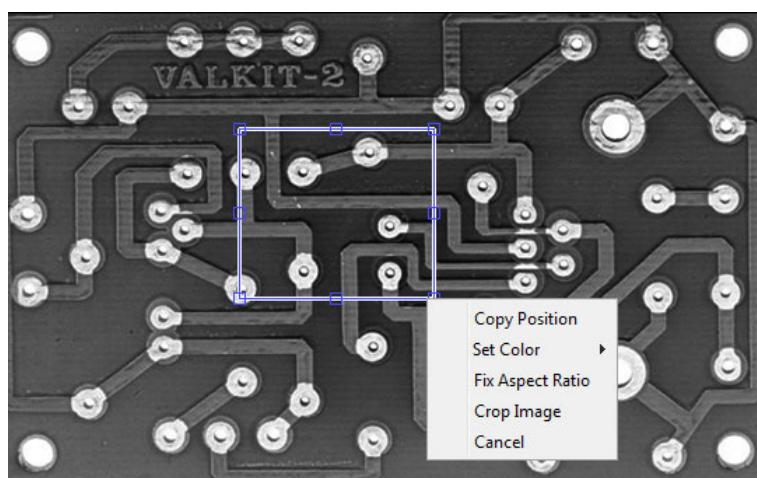


Figura f.252 - Herramienta de recorte de la imagen

La siguiente tabla describe las diferentes acciones posibles en la herramienta:

Acción	Operación
Cancelar lo hecho por la herramienta.	Pulsar Retroceso , Escape o Suprimir , o hacer clic derecho dentro del rectángulo de selección y pulsar Cancel dentro del menú contextual. Nota: Al eliminar/cancelar la herramienta, la función devuelve un valor vacío.
Redimensionar la selección de la imagen.	Poner el ratón encima de las líneas del rectángulo de selección, el cursor cambiará de forma, hacer clic y desplazar para modificar la selección.
Mover el rectángulo de selección.	Mover el ratón dentro de la zona de selección hasta que el puntero cambie a una forma de cruz, y mover el rectángulo pinchando y desplazando.
Cambiar el color del rectángulo de selección.	Hacer clic derecho dentro del rectángulo y luego pulsar SetColor (“Elegir Color”) del menú contextual.
Recortar la imagen.	Hacer doble clic con el botón izquierdo o un solo clic con el botón derecho del ratón y luego seleccionar Crop Image (“Recortar imagen”) en el menú contextual.
Recibir las coordenadas del rectángulo de selección.	Hacer clic derecho dentro del rectángulo de selección y seleccionar Copy Position (“Copiar posición”) en el menú contextual. <code>imcrop</code> copia al portapapeles un vector de posición de cuatro elementos (<code>[xmin ymin width height]</code>).

`I2 = imcrop(I)` muestra la imagen `I` en una nueva ventana y crea la herramienta de recorte asociada con la imagen. `I` puede ser una imagen en escala de grises, una imagen de color verdadero, o un array tipo `logical`. La imagen recortada devuelta, `I2`, es del mismo tipo que `I`.

`X2 = imcrop(X, map)` muestra la imagen indexada `X` en una nueva ventana utilizando el mapa de color `map` y creando la herramienta de recorte asociada con la imagen.

`I = imcrop(h)` crea la herramienta de recorte asociada a la imagen apuntada por el puntero `h`. `h` puede ser una imagen, eje, o puntero a una figura. Si `h` es un eje o puntero a una figura, la herramienta de selección actúa sobre la primera imagen encontrada en el objeto contenedor.

Nota: Con estas sintaxis, la línea de comandos de MATLAB se bloquea hasta que se complete la acción con la herramienta.

`I2 = imcrop(I, rect)` recorta la imagen `I`. `rect` es un vector de posición de cuatro elementos `[xmin ymin width height]` que especifica el tamaño y posición del rectángulo de selección.

`X2 = imcrop(X, map, rect)` recorta la imagen indexada `X`. `map` especifica el mapa de color utilizado con la imagen `X`. `rect` es un vector de posición de cuatro elementos `[xmin ymin width height]` que especifica el tamaño y posición del rectángulo de selección.

`[...] = imcrop(x, y,...)` especifica un sistema de coordenadas no por defecto para la imagen destino. `x` e `y` son vectores de dos elementos que especifican las propiedades `XData` e `YData`.

`[I2 rect] = imcrop(...)` devuelve el rectángulo de selección en la variable `rect`, que es un vector de posición de cuatro elementos.

`[X,Y,I2,rect] = imcrop(...)` devuelve `x` e `y`, vectores de dos elementos que especifican las propiedades `XData` e `YData` de la imagen destino.

Si se especifica `rect` como argumento de entrada, la imagen de entrada puede ser de tipo `numeric` o `logical` y debe ser real y no disperso. `rect` es de clase `double`.

Si no se especifica `rect` como argumento de entrada, `imcrop` llama a `imshow`. `imshow` espera que `I` sea de clase `logical`, `uint8`, `uint16`, `int16`, `single` o `double`. La imagen en color verdadero puede ser `uint8`, `int16`, `uint16`, `single` o `double`. `X` puede ser `logical`, `uint8`, `uint16`, `single` o `double`. La imagen de entrada debe ser real y no dispersa.

Si se especifica una imagen como argumento de entrada, la imagen de salida será de la misma clase que la de entrada.

Si no se especifica una imagen como argumento de entrada, por ejemplo llamando a la función `imcrop` con ningún argumento de entrada o con un puntero, la imagen de salida será de la misma clase que la de entrada salvo para los tipos `int16` o `single`, en los que la imagen de salida será de clase `double`.

Nota: `rect` es un vector de posición de cuatro elementos [`xmin` `ymin` `width` `height`] que especifica el tamaño y posición del rectángulo de selección. Como `rect` se define en términos de coordenadas espaciales, las variables `width` y `height` no siempre se corresponden exactamente con el tamaño de la imagen de salida. Por ejemplo, si `rect` vale [20 20 40 30], utilizando el sistema de coordenadas por defecto, la esquina superior izquierda del rectángulo especificado es el centro del pixel (20,20) y la esquina inferior derecha es el centro del pixel (50,60). La imagen resultante de salida es de tamaño 31-por-41, no de 30-por-40, porque la imagen de salida incluye todos los píxeles de la imagen de entrada que están parcial o completamente encerrados por el rectángulo.

Ejemplos de uso

Ejemplo 1: Recortar una imagen y mostrar el recorte.

```
I = imread('circuito.tif');
I2 = imcrop(I,[75 68 130 112]);
imshow(I)
figure, imshow(I2)
```

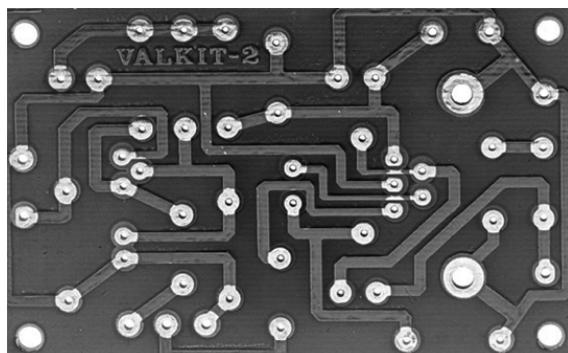


Figura f.249 - Imagen original (I) sin procesar



Figura f.253 - Imagen recortada (I2)

Funciones relacionadas

`imrect`, `zoom`

imdilate

Dilatación de una imagen

Sintaxis

```
IM2 = imdilate(IM,SE)
IM2 = imdilate(IM,NHOOD)
IM2 = imdilate(IM,SE,PACKOPT)
IM2 = imdilate(...,SHAPE)
```

Descripción

`IM2 = imdilate(IM,SE)` dilata la imagen en escala de grises, binaria o binaria empaquetada `IM` y devuelve la imagen dilatada `IM2`. El argumento `SE` es un elemento estructurante (o un array de elementos estructurantes) como el devuelto por la función `strel`.

Si `IM` es de clase `logical` y el elemento estructurante es plano, `imdilate` realiza una dilatación binaria; de cualquier otro modo, realiza una dilatación en escala de grises. Si `SE` es un array de elementos estructurantes, `imdilate` realiza dilataciones múltiples de la imagen de entrada, utilizando sucesivamente cada elemento estructurante en `SE`.

`IM2 = imdilate(IM,NHOOD)` dilata la imagen `IM`, donde `NHOOD` es una matriz de 0's y 1's que especifica la vecindad o patrón del elemento estructurante. Esto es equivalente a la expresión `imdilate(IM,strel(NHOOD))`. La función `imdilate` determina el elemento central de la vecindad como `floor((size(NHOOD)+1)/2)`.

`IM2 = imdilate(IM,SE,PACKOPT)` o `imdilate(IM,NHOOD,PACKOPT)` especifica si `IM` es una imagen binaria empaquetada en la variable `PACKOPT` que puede tener cualquiera de los siguientes valores:

Valor	Descripción
'ispacked'	<code>IM</code> se trata como una imagen binaria empaquetada igual que la producida por <code>bwpack</code> . <code>IM</code> debe ser un array bidimensional de clase <code>uint32</code> y <code>SE</code> debe ser un elemento estructurante plano bidimensional. Si el valor de <code>PACKOPT</code> es ' <code>ispacked</code> ', <code>PADOPT</code> debe ser ' <code>same</code> '.
'notpacked'	<code>IM</code> se trata como si fuera un array normal. (Por defecto)

`IM2 = imdilate(...,SHAPE)` especifica el tamaño de la imagen de salida en la variable `SHAPE`, que puede tener cualquiera de los siguientes valores:

Valor	Descripción
'same'	Crea la imagen de salida con el mismo tamaño que la de entrada. Si el valor de <code>PACKOPT</code> es 'ispacked', <code>PADOPT</code> debe ser 'same'. (Por defecto)
'full'	Calcula una dilatación completa.

`IM` puede ser de clase `logical` o `numeric` y debe ser real y no disperso. Puede tener cualquier dimensión. Si `IM` es de clase `logical`, `SE` debe ser plano. La salida será de la misma clase que la entrada. Si la entrada es binaria empaquetada, entonces la salida será del mismo tipo.

Ejemplos de uso

Ejemplo 1: Dilatar una imagen binaria con una línea vertical como elemento estructurante.

```
I = imread('texto.tif');
se = strel('line',11,90);
I2 = imdilate(I,se);
imshow(bw), title('Original')
figure, imshow(I2), title('Dilated')
```



Figura f.80 - Imagen original (I)



Figura f.254 - Imagen dilatada (I2) con imdilate

Ejemplo 2: Dilatar una imagen en escala de grises con una bola que rueda como elemento estructurante (no se puede dilatar una imagen binaria con un elemento estructurante no plano).

```
I = imread('cameraman.tif');
se = strel('ball',5,5);
I2 = imdilate(I,se);
imshow(I)
figure, imshow(I2)
```



Figura f.179 - Imagen original (I)

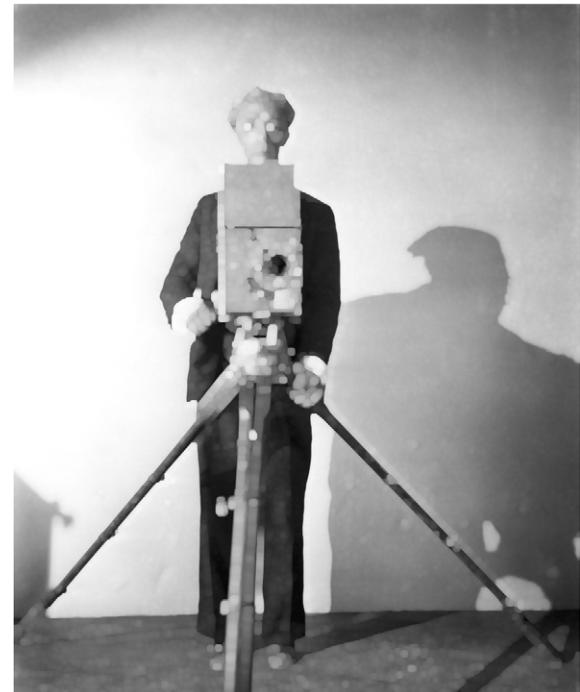


Figura f.255 - Imagen dilatada (I2) con imdilate

Funciones relacionadas

bwpack, bwunpack, conv2, filter2, imclose, imerode, imopen, strel

imdisplayrange

Herramienta para mostrar el rango de visualización de una imagen

Sintaxis

```
imdisplayrange  
imdisplayrange(h)  
imdisplayrange(hparent,himage)  
hpanel = imdisplayrange(...)
```

Descripción

`imdisplayrange` crea una herramienta en la figura actual que muestra el rango de visualización de la imagen o imágenes de la figura.

La herramienta es un objeto tipo `uipanel`, posicionado en la esquina inferior derecha de la figura, que muestra el rango de visualización con la cadena de texto `Display range:` (“Rango de visualización”) seguida de los valores correspondientes al rango.

Para imágenes indexadas, binarias o de color verdadero, el rango de visualización no es aplicable y se muestra vacío (`[]`).

`imdisplayrange(h)` crea la herramienta para mostrar el rango en la figura especificada por el puntero `h`, donde `h` es un puntero a una imagen, eje, objeto `uipanel` o figura. Los ejes, objetos `uipanel` o figuras deben contener al menos una imagen.

`imdisplayrange(hparent,himage)` crea la herramienta para mostrar el rango donde apunta `hparent` y muestra el rango de la imagen apuntada por `himage`. `himage` es un puntero a una imagen o array de punteros a imágenes. `hparent` es un puntero a la figura u objeto `uipanel` que contiene la herramienta.

`hpanel = imdisplayrange(...)` devuelve un puntero al objeto con la herramienta que muestra el rango de visualización.

Nota: para mostrar el rango de visualización de una imagen la herramienta puede trabajar con múltiples imágenes en una figura pero se tendrá que poner el cursor encima de cada una de las imágenes de la figura para mostrar el rango de cada una de ellas. Cuando no se posiciona el cursor encima de ninguna imagen la herramienta muestra el rango `[black,white]`.

Ejemplos de uso

Ejemplo 1: Mostrar una imagen y su rango de visualización.

```
imshow('cameraman.tif')  
imdisplayrange;
```

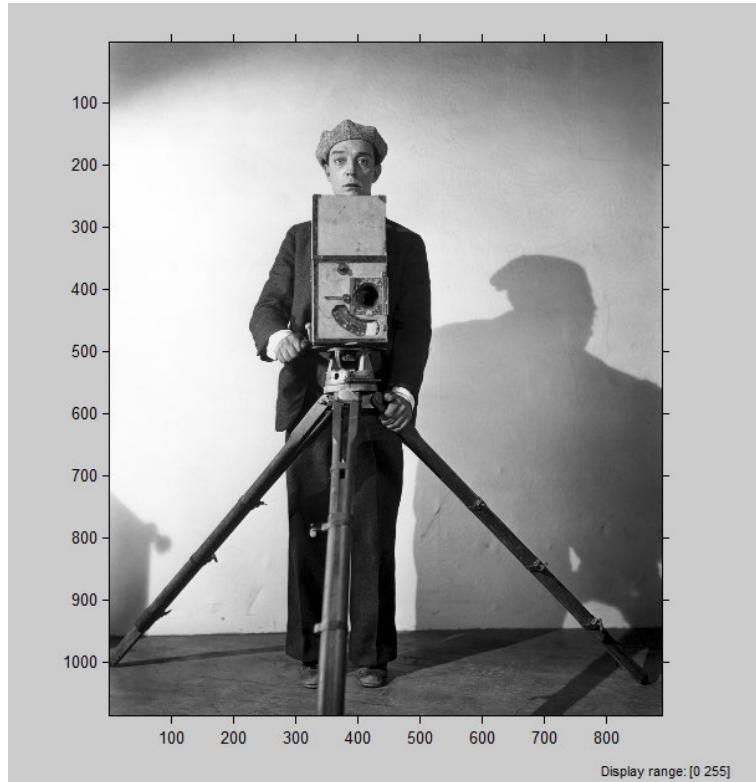


Figura f.256 - Imagen con información de rango

Funciones relacionadas

```
imtool
```

imdistline

Herramienta de medida de distancia

Sintaxis

```
h = imdistline  
h = imdistline(hparent)  
h = imdistline(..., x, y)
```

Descripción

`h = imdistline` crea una herramienta de medida de distancia en los ejes actuales. La función devuelve `h`, un puntero a un objeto tipo `imdistline`.

La herramienta de medida de distancia consiste en una línea desplazable, que se superpone con los ejes, y que mide la distancia entre sus dos puntos finales. La herramienta muestra la distancia medida en un texto sobreimpresionado encima de la línea. La herramienta especifica la distancia en la unidad de medida definida en las propiedades `xData` e `yData` y que es de píxeles por defecto.

Para mover la herramienta se debe posicionar el ratón encima de la línea hasta que el cursor cambie de forma y después hacer clic y desplazar. Para modificar el tamaño de la línea se debe de mover el puntero del ratón a cualquiera de los dos puntos finales de la línea, hacer clic y desplazar el punto. También se pueden controlar distintos aspectos del funcionamiento y de la apariencia de la herramienta a través del menú contextual que aparece haciendo clic derecho con el ratón encima de la línea de la herramienta.

`h = imdistline(hparent)` crea una herramienta de medida de la distancia en el objeto especificado por `hparent`, típicamente un objeto con ejes pero puede ser cualquier objeto del tipo `hggroup`.

`h = imdistline(..., x, y)` crea una herramienta de medida de la distancia con los puntos finales de la línea localizados en las posiciones especificadas por los vectores `x` e `y`, donde `x = [x1 x2]` e `y = [y1 y2]`.

Las funciones del menú contextual de la herramienta se describen a continuación:

Acción	Operación
Exportar los datos de final de línea y distancia al espacio de trabajo.	Seleccionar Export to Workspace
Mostrar u ocultar la etiqueta de distancia.	Seleccionar Show Distance Label
Especificar restricciones para el desplazamiento horizontal y vertical.	Seleccionar Constrain Drag
Cambiar el color de la línea.	Seleccionar Set Color
Eliminar el objeto de distancia.	Seleccionar Delete

La herramienta de medida de distancia contiene una API (*Application Programming Interface*, Interfaz de programación de aplicaciones) o conjunto de funciones internas que se pueden utilizar para obtener y controlar diferentes aspectos de la herramienta. Para poder utilizar estas funciones se debe recibir la estructura asociada a las funciones utilizando la función `iptgetapi` a la que se le pasa un puntero `h` a la herramienta de medida de distancia de la siguiente manera:

```
api = iptgetapi(h)
```

La siguiente tabla muestra las funciones disponibles en la API de la herramienta:

Función	Descripción
<code>getDistance</code>	Devuelve la distancia entre los extremos de la línea de la herramienta. <code>dist = getDistance()</code> El valor devuelto está en las unidades definidas por las propiedades XData e YData y será en píxeles por defecto.
<code>getAngleFromHorizontal</code>	Devuelve el ángulo en grados entre la línea definida por la herramienta y el eje horizontal. El ángulo devuelto estará entre 0 y 180 grados. <code>angle = getAngleFromHorizontal()</code>
<code>setLabelTextFormatter</code>	Especifica la cadena de texto utilizada para mostrar la distancia de un modo esperado por la función <code>sprintf</code> . <code>setLabelTextFormatter(str)</code>
<code>getLabelTextFormatter</code>	Recoge un array de caracteres que especifica la cadena de texto utilizada para mostrar la distancia de un modo esperado por la función <code>sprintf</code> .

	<code>str = getLabelTextFormatter()</code>
<code>setLabelVisible</code>	Especifica la visibilidad del texto de la herramienta. <code>setLabelVisible(h,TF)</code> h es la herramienta de medida de distancia. TF es un escalar lógico. Cuando el texto de la herramienta es visible, TF es verdadero. Cuando es invisible, TF es falso.
<code>getLabelVisible</code>	Lee la visibilidad del texto de la herramienta. <code>TF = getLabelVisible(h)</code> h es la herramienta de medida de distancia. TF es un escalar lógico. Cuando el texto de la herramienta es visible, TF es verdadero. Cuando es invisible, TF es falso.
<code>setPosition</code>	Especifica la posición de los puntos finales de la línea de la herramienta. <code>setPosition(X,Y)</code> <code>setPosition([X1 Y1; X2 Y2])</code>
<code>getPosition</code>	Devuelve la posición de los puntos finales de la línea de la herramienta. <code>pos = getPosition()</code> pos es un array de 2-por-2 [X1 Y1; X2 Y2].
<code>delete</code>	Elimina la herramienta asociada con la API. <code>delete()</code>
<code>setColor</code>	Especifica el color utilizado en la línea de la herramienta. <code>setColor(new_color)</code> new_color puede ser un vector de tres elementos que especifica un color RGB, o una cadena de texto que especifica los nombres de un color predefinido, como el blanco con 'white' o 'w'.
<code>getColor</code>	Lee el color utilizado para dibujar el objeto h. <code>color = getColor(h)</code> color es un vector de tres elementos que especifica un color RGB.
<code>addNewPositionCallback</code>	Añade el puntero a la función fcn a la lista de funciones de llamada o 'callback' de nueva posición (se las llamará cada vez que se cambie la posición del objeto con la posición como argumento). <code>id = addNewPositionCallback(fcn)</code> Siempre que la herramienta cambie su posición, se llamará a cada función en la lista con la siguiente sintaxis: <code>fcn(pos)</code> pos es un array de 2-por-2 [X1 Y1; X2 Y2].

	El valor devuelto <code>id</code> , se utiliza solo con <code>removeNewPositionCallback</code> .
<code>removeNewPositionCallback</code>	Elimina la función correspondiente de la lista de funciones de llamada o ‘callback’ de nueva posición. <code>removeNewPositionCallback(id)</code> <code>id</code> es el identificador devuelto por <code>addNewPositionCallback</code> .
<code>setPositionConstraintFcn</code>	Hace que la función de restricción de posición sea la especificada por el puntero <code>fcn</code> . Utilizar esta función para controlar dónde puede moverse y redimensionarse la herramienta. <code>setPositionConstraintFcn(fcn)</code> Siempre que la herramienta se mueva o se redimensione, se llamará a la función de restricción con la siguiente sintaxis. <code>constrained_position = fcn(new_position)</code> <code>new_position</code> es un array de 2-por-2 [X1 Y1; X2 Y2].
<code>getPositionConstraintFcn</code>	Devuelve el puntero a la función de restricción actual. <code>fcn = getPositionConstraintFcn()</code>

Ejemplos de uso

Ejemplo 1: Insertar la herramienta de medida de distancia en una imagen. Utilizar la función `makeConstrainToRectFcn` para especificar una función de restricción de desplazamiento que evite desplazar la herramienta fuera de los límites de la imagen.

```

figure, imshow('cameraman.tif')
h = imdistline(gca);

% Accedemos a la API de la herramienta.

api = iptgetapi(h);
fcn = makeConstrainToRectFcn('imline',get(gca,'XLim'),get(gca,'YLim'));
api.setDragConstraintFcn(fcn);

```

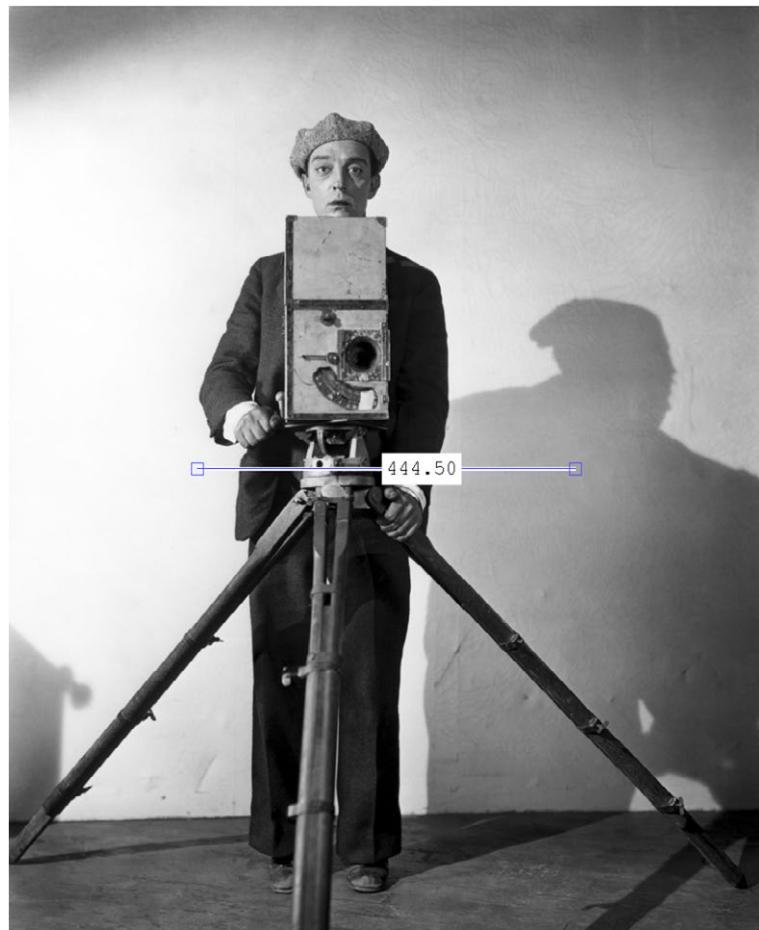


Figura f.257 - Imagen con la herramienta de medida de distancia

Funciones relacionadas

`iptgetapi`, `makeConstrainToRectFcn`

imdivide

División de una imagen entre otra o entre una constante

Sintaxis

```
Z = imdivide(X,Y)
```

Descripción

`Z = imdivide(X,Y)` divide cada elemento del array `X` entre el elemento correspondiente del array `Y` y devuelve el resultado en el array de salida `Z`. `X` e `Y` son arrays numéricos reales, no dispersos y siempre con el mismo tamaño y clase salvo cuando `Y` es un escalar tipo `double`. `Z` tendrá el mismo tipo y clase que `X` e `Y`.

Si `X` es un array de enteros, se truncarán los elementos en la salida que excedan el rango de los enteros, y se redondearán los valores fraccionarios.

Ejemplos de uso

Ejemplo 1: Dividir dos arrays de clase `uint8`. Notar que los valores fraccionarios se redondean al entero más cercano.

```
X = uint8([ 255 10 75; 44 225 100]);
Y = uint8([ 50 20 50; 50 50 50 ]);
Z = imdivide(X,Y)
```

```
Z =
      5       1       2
      1       5       2
```

Ejemplo 2: Dividir una imagen por una constante.

```
I = imread('circuito.tif');
J = imdivide(I,2);
imshow(I)
figure, imshow(J)
```

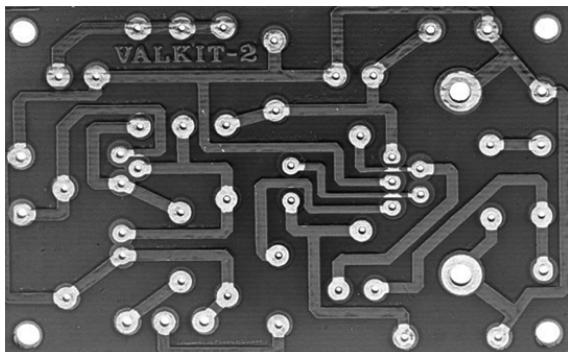


Figura f.249 - Imagen original (I) sin procesar

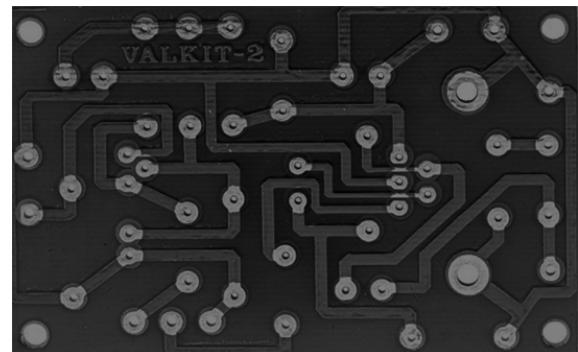


Figura f.258 - Imagen dividida por constante (J)

Ejemplo 3: Dividir una imagen por su fondo estimado.

```
I = imread('circuito.tif');
background = imopen(I,strel('disk',15));
J = imdivide(I,background);
imshow(J,[])
figure, imshow(I)
```

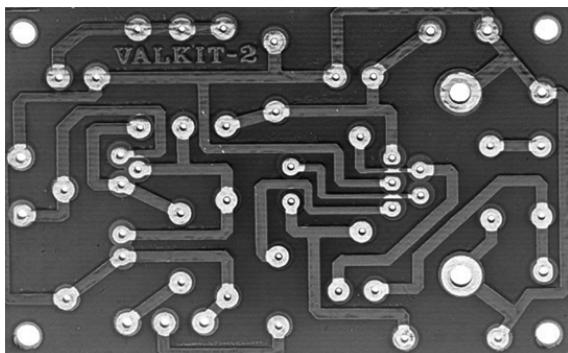


Figura f.249 - Imagen original (I) sin procesar

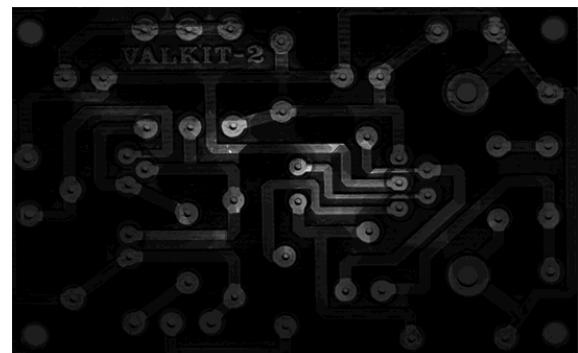


Figura f.259 - Imagen dividida por fondo (J)

Funciones relacionadas

imabsdiff, imadd, imcomplement, imlincomb, immultiply, imsubtract, ippl

imellipse

Creación de una elipse desplazable

Sintaxis

```
h = imellipse
h = imellipse(hparent)
h = imellipse(hparent, position)
H = imellipse(...,param1, val1, ...)
```

Descripción

`h = imellipse` posiciona una elipse desplazable en los ejes actuales. La función devuelve `h`, un puntero a un objeto `imellipse`. La elipse tiene un menú contextual asociado con ella que controla aspectos de su visualización y comportamiento y que se activa haciendo clic derecho sobre su contorno.

`h = imellipse(hparent)` posiciona una elipse desplazable en el objeto especificado por `hparent`, que normalmente es un eje pero puede ser cualquier otro objeto perteneciente al grupo `hggroup`.

`h = imellipse(hparent, position)` posiciona una elipse desplazable en objeto especificado por `hparent`. `position` es un vector de cuatro elementos que especifica la localización inicial de la elipse en términos de un rectángulo circundante. `position` tiene la estructura `[xmin ymin width height]`.

`H = imellipse(...,param1, val1, ...)` crea una elipse desplazable, especificando parámetros sus valores correspondientes para controlar el comportamiento de la elipse. La siguiente tabla muestra los parámetros posibles. Los nombres de los parámetros se pueden abbreviar y no son sensibles a las mayúsculas:

Parámetro	Valor
'PositionConstraintFcn'	Puntero a una función de restricción de posición que se llama cuando se desplaza el ratón. Se puede usar esta herramienta para desplazar la elipse. Si se utiliza <code>imellipse</code> en una imagen y no se especifica una función de restricción de posición, los usuarios podrán desplazar la elipse fuera de los límites de la imagen y

	perder la elipse. Cuando los ejes se crean con la función <code>plot</code> se expanden automáticamente para acomodar el movimiento de la elipse.
--	---

Cuando se llama a la función `imellipse` con una sintaxis interactiva, el puntero del ratón tendrá la forma de una cruz cuando se esté encima de la imagen. Se deberá hacer clic y desplazar el ratón para crear la elipse con el tamaño deseado. Una vez creada, se podrá desplazar poniendo el cursor encima de ella, haciendo clic y desplazando. Si se hace clic derecho encima de ella aparecerá el menú contextual que permite seleccionar diferentes opciones de visualización y comportamiento.

La siguiente tabla muestra las acciones interactivas que se pueden realizar con la herramienta `imellipse`.

Acción	Operación
Mover la elipse.	Mover el puntero del ratón dentro de la elipse. El puntero cambiará a forma de cruz. Hacer clic y desplazar el ratón para mover la elipse.
Redimensionar la elipse.	Mover el puntero del ratón encima de los cuadrados de redimensionamiento de la elipse. El puntero tendrá forma de flecha. Hacer clic y desplazar el ratón para mover la elipse.
Cambiar el color de la elipse.	Mover el puntero del ratón dentro de la elipse. Hacer clic derecho y seleccionar Set Color (“Establecer color”) del menú contextual.
Leer la posición actual de la elipse.	Mover el puntero del ratón dentro de la elipse. Hacer clic derecho y seleccionar Copy Position (“Copiar posición”) del menú contextual. <code>imellipse</code> copia al portapapeles un vector de posición de cuatro elementos [xmin ymin width height].
Mantener las proporciones de la elipse al redimensionarla.	Mover el puntero del ratón dentro de la elipse. Hacer clic derecho y seleccionar Fix Aspect Ratio (“Mantener proporciones”) del menú contextual.

Cada objeto `imellipse` soporta una serie de métodos que pueden verse mediante el comando `methods imellipse` y que se listan a continuación:

`addNewPositionCallback`

Añade una función a la lista de funciones de llamada o ‘callback’ de nueva posición (se las llamará cada vez que se cambie la posición del objeto ROI (*Region-of-interest*) con la posición como argumento).

`createMask`

Crea una máscara dentro de la imagen.

`delete`

Elimina el objeto ROI.

`getColor`

Lee el color utilizado para dibujar el objeto ROI.

`getPosition`

Lee la posición actual de la elipse.

`getPositionConstraintFcn`

Devuelve el puntero a la función de restricción de posición actual.

`getVertices`

Devuelve los vértices de la elipse.

`removeNewPositionCallback`

Elimina la función de rellamada o ‘callback’ de nueva posición del objeto ROI.

`resume`

Continúa la ejecución de la línea de comandos.

`setColor`

Especifica el color utilizado para dibujar el objeto ROI.

`setConstrainedPosition`

Especifica la nueva posición del objeto ROI.

`setFixedAspectRatioMode`

Mantiene las proporciones cuando se redimensiona el objeto.

`setPosition`

Especifica la nueva posición de la elipse.

`setPositionConstraintFcn`

Especifica la función de restricción de posición del objeto ROI.

`setResizable`

Activa el redimensionado de la elipse.

`wait`

Bloquea la ejecución de la línea de comandos hasta que se finaliza el posicionamiento del objeto ROI.

Los comandos anteriores se ejecutan sobre `h`, el puntero al objeto `imellipse`.

Ejemplos de uso

Ejemplo 1: Crear una elipse utilizando rellamadas o ‘callbacks’ para visualizar su posición actualizada en el título de la figura. El ejemplo muestra cómo utilizar la función `makeConstrainToRectFcn` para mantener la elipse en los límites de rango `xlim` e `ylim`.

```
figure, imshow('cameraman.tif')

% Creación de la elipse en la posición especificada.

h = imellipse(gca, [10 10 200 200]);

% Añadir la función título a la lista de funciones de rellamada para
% nuevas posiciones, para que al mover la elipse a una nueva posición se
% llame a dicha función título con la posición como argumento, lo que hará
% mostrar en el objeto la posición como título.

addNewPositionCallback(h,@(p) title(mat2str(p,3)));

% Restringir el rango de desplazamiento de la elipse a los límites.

fcn =
makeConstrainToRectFcn('imellipse',get(gca,'XLim'),get(gca,'YLim'));
setPositionConstraintFcn(h,fcn);
```

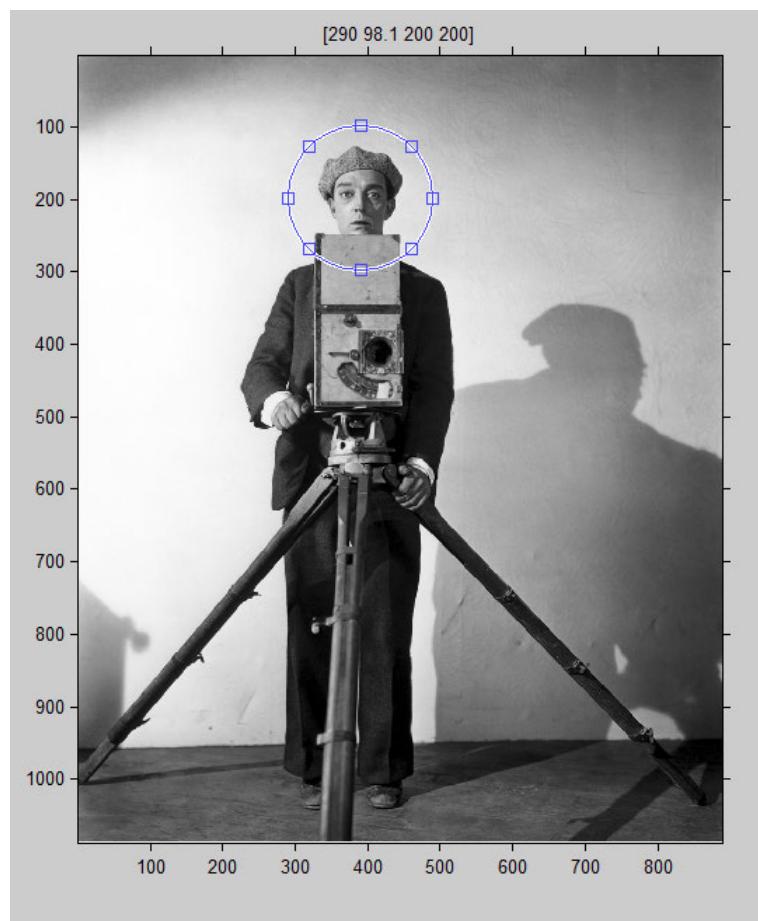


Figura f.260 - Imagen con elipse y su posición

Funciones relacionadas

imfreehand, imline, impoint, impoly, imrect, imroi, iptgetapi,
makeConstrainToRectFcn

imerode

Erosión de una imagen

Sintaxis

```
IM2 = imerode(IM,SE)
IM2 = imerode(IM,NHOOD)
IM2 = imerode(...,PACKOPT,M)
IM2 = imerode(...,SHAPE)
```

Descripción

`IM2 = imerode(IM,SE)` erosiona la imagen en escala e grises, binaria o binaria empaquetada `IM`, devolviendo la imagen erosionada `IM2`. El argumento `SE` es un elemento estructurante o un array de elementos estructurantes devueltos por la función `strel`.

Si `IM` es lógica y el elemento estructurante es plano, `imerode` realiza una erosión binaria; de otra manera realiza una erosión en escala de grises. Si `SE` es un array de elementos estructurantes, `imerode` realiza múltiples erosiones de la imagen de entrada utilizando sucesivamente cada elemento estructurante de `SE`.

`IM2 = imerode(IM,NHOOD)` erosiona la imagen `IM`, donde `NHOOD` es un array de 0's y 1's que especifica la vecindad o patrón del elemento estructurante. Esto es equivalente a la sintaxis `imerode(IM,strel(NHOOD))`. La función `imerode` determina el elemento central de la vecindad mediante `floor((size(NHOOD)+1)/2)`.

`IM2 = imerode(...,PACKOPT,M)` especifica si `IM` es una imagen binaria empaquetada y si lo es, proporciona la dimensión o fila `M` de la imagen original desempaquetada. `PACKOPT` puede ser cualquiera de los siguientes valores:

Valor	Descripción
'ispacked'	<code>IM</code> se trata como una imagen binaria empaquetada, como la producida con la función <code>bwpack</code> . <code>IM</code> debe ser un array bidimensional de clase <code>uint32</code> y <code>SE</code> debe ser un elemento estructurante plano bidimensional. Se debe especificar un valor para <code>M</code> .
'notpacked'	<code>IM</code> se trata como un array normal. (Por defecto)

`IM2 = imerode(. . . , SHAPE)` especifica el tamaño de la imagen de salida. `SHAPE` puede ser cualquiera de los siguientes valores:

Valor	Descripción
{ 'same' }	Crea la imagen de salida con el mismo tamaño que la de entrada. Si el valor de <code>PACKOPT</code> es ' <code>ispacked</code> ', <code>SHAPE</code> debe valer ' <code>same</code> '.
'full'	Calcula la erosión completa.

`IM` puede ser numérica o lógica y puede ser de cualquier dimensión. Si `IM` es lógica y el elemento estructurante es plano, la imagen de salida será lógica. De otro modo la imagen de salida será de la misma clase que la imagen de entrada.

Ejemplos de uso

Ejemplo 1: Erosionar una imagen binaria con un disco como elemento estructurante.

```
I = imread('circulos.tif');
se = strel('disk',11);
E = imerode(I,se);
imshow(I), figure, imshow(E)
```

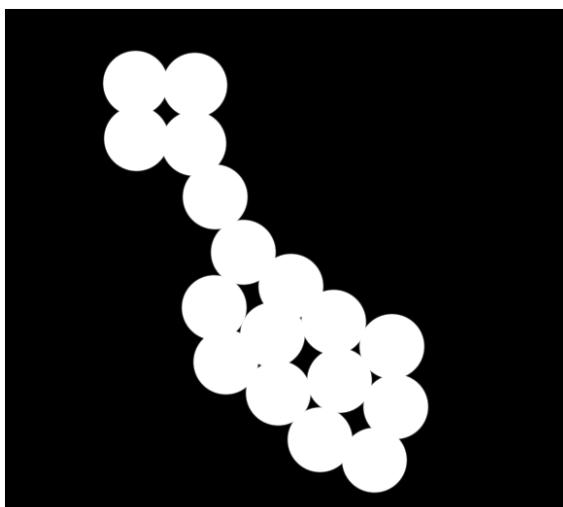


Figura f.245 - Imagen original (`I`) sin procesar

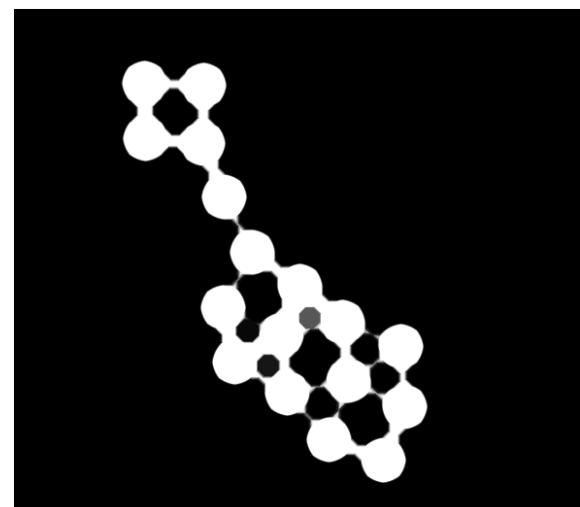


Figura f.261 - Imagen erosionada (`E`) con `imerode`

Ejemplo 2: Erosionar una imagen en escala de grises con una “bola que rueda” como elemento estructurante.

```
I = imread('cameraman.tif');
se = strel('ball',5,5);
E = imerode(I,se);
imshow(I)
figure, imshow(E)
```



Figura f.179 - Imagen original (I) sin procesar



Figura f.262 - Imagen erosionada (E) con imerode

Funciones relacionadas

`bwpack`, `bwunpack`, `conv2`, `filter2`, `imclose`, `imdilate`, `imopen`, `strel`

imextendedmax

Transformada extendida máxima

Introducción

La *toolbox* para el procesado de imágenes en MATLAB incluye varias funciones que se pueden utilizar para encontrar áreas de mayor o menor intensidad en una imagen. La Transformada extendida máxima es una de esas funciones y es útil para identificar grupos de píxeles que tienen un valor de intensidad significativamente más alto que el de sus vecinos, o lo que es lo mismo, píxeles que son máximos regionales. Es por lo tanto una función útil en técnicas de segmentación de imágenes.

Un máximo (o mínimo) regional son un conjunto de píxeles conectados con un valor de intensidad constante y cuyos vecinos tienen un valor inferior (o mayor). Por ejemplo, la siguiente imagen cuenta con dos máximos regionales principales (bloques de píxeles de valor 13 y 18) y varios máximos regionales inferiores de valor 11.

```
A =
10 10 10 10 10 10 10 10 10 10
10 13 13 13 10 10 11 10 11 10
10 13 13 13 10 10 10 11 10 10
10 13 13 13 10 10 11 10 11 10
10 10 10 10 10 10 10 10 10 10
10 11 10 10 10 18 18 18 10 10
10 10 10 11 10 18 18 18 10 10
10 10 11 10 10 18 18 18 10 10
10 11 10 11 10 10 10 10 10 10
10 10 10 10 10 10 11 10 10 10
```

Las funciones `imregionalmax` e `imregionalmin` identifican todos los máximos o mínimos regionales respectivamente produciendo una imagen binaria de salida. Para la imagen anterior, la imagen de salida tras aplicar la función de regional máxima sería la siguiente:

```
B = imregionalmax(A)
```

```
B =
0 0 0 0 0 0 0 0 0 0
0 1 1 1 0 0 1 0 1 0
0 1 1 1 0 0 0 1 0 0
0 1 1 1 0 0 0 1 0 1
0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 1 1 1 0 0
0 0 0 1 0 1 1 1 0 0
0 0 1 0 0 1 1 1 0 0
0 1 0 1 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0
```

Para identificar áreas de una imagen en donde el cambio de intensidad sea extremo, tal que la diferencia entre el pixel y los píxeles vecinos sea mayor (o menor) que un cierto valor de umbral especificado, se deben usar las funciones `imextendedmax` e `imextendedmin` (Transformada extendida máxima y mínima). Por ejemplo, para la imagen anterior, la imagen binaria de salida para un umbral de 4 sería la siguiente:

```
B = imextendedmax(A, 4)
```

```
B =
 0   0   0   0   0   0   0   0   0   0
 0   0   0   0   0   0   0   0   0   0
 0   0   0   0   0   0   0   0   0   0
 0   0   0   0   0   0   0   0   0   0
 0   0   0   0   0   0   0   0   0   0
 0   0   0   0   0   0   0   0   0   0
 0   0   0   0   0   1   1   1   0   0
 0   0   0   0   0   1   1   1   0   0
 0   0   0   0   0   1   1   1   0   0
 0   0   0   0   0   0   0   0   0   0
 0   0   0   0   0   0   0   0   0   0
```

Si además se quiere limitar la altura de los máximos o la profundidad de los mínimos regionales a un cierto valor, se usan las funciones `imhmax` e `imhmin` (Transformada H-máxima y H-mínima) que eliminan cualquier máximo o mínimo regional en donde la diferencia entre el pixel regional y los píxeles vecinos sea menor (o mayor) que un cierto valor de umbral. Por ejemplo, para la imagen original `A`, la imagen de salida tras aplicar la Transformada H-máxima con la función `imhmax` utilizando un umbral de valor 4 sería la siguiente:

```
B = imhmax(A, 4)
```

```
B =
 10   10   10   10   10   10   10   10   10   10
 10   10   10   10   10   10   10   10   10   10
 10   10   10   10   10   10   10   10   10   10
 10   10   10   10   10   10   10   10   10   10
 10   10   10   10   10   10   10   10   10   10
 10   10   10   10   10   10   10   10   10   10
 10   10   10   10   10   14   14   14   10   10
 10   10   10   10   10   14   14   14   10   10
 10   10   10   10   10   14   14   14   10   10
 10   10   10   10   10   10   10   10   10   10
 10   10   10   10   10   10   10   10   10   10
```

Se puede observar que la transformada extendida máxima es el máximo regional de la transformada H-máxima.

Sintaxis

```
BW = imextendedmax(I,H)
BW = imextendedmax(I,H,conn)
```

Descripción

`BW = imextendedmax(I,H)` calcula la transformada extendida máxima de la imagen `I`. `H` es un escalar no negativo que especifica el umbral o diferencia que debe existir entre los píxeles máximos regionales y los píxeles vecinos. Es una forma de identificar grupos de píxeles que tienen un valor de intensidad significativamente más alto que el de sus vecinos.

`imextendedmax` utiliza por defecto vecindades de 8 píxeles para imágenes en 2-D y vecindad de 26 píxeles para imágenes en 3-D. Para dimensiones mayores, `imextendedmax` utiliza la conectividad dada por `conndef(ndims(I), 'maximal')`.

`BW = imextendedmax(I,H,conn)` calcula la transformada extendida máxima donde `conn` especifica la conectividad y puede tener cualquiera de los siguientes valores escalares (siempre simétrica respecto su elemento central):

Valor	Significado
Conectividades Bidimensionales	
4	Vecindad de 4 píxeles: Un par de píxeles colindantes forman parte del mismo objeto solo si están activos y conectados en vertical u horizontal.
8	Vecindad de 8 píxeles: Un par de píxeles colindantes forman parte del mismo objeto solo si están activos. (Por defecto)
Conectividades Tridimensionales	
6	Vecindad de 6 píxeles.
18	Vecindad de 18 píxeles.
26	Vecindad de 26 píxeles. (Por defecto)
<p>Nota: La conectividad se puede definir para cualquier dimensión de forma más general utilizando para <code>conn</code> una matriz de 3-por-3-por-...-por-3 de ceros y unos. Los unos definen las localizaciones de los píxeles vecinos relativos al elemento</p>	

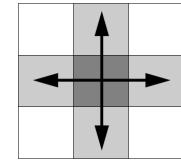


Figura f.32 - vecindad 4

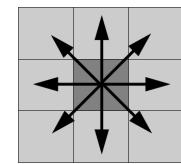


Figura f.33 - vecindad 8

central de conn. Por ejemplo, conn = ones(3) define una conectividad de 8 píxeles.

I puede ser de cualquier clase numérica no dispersa y de cualquier dimensión. BW tiene el mismo tamaño que I y es siempre de tipo lógico.

Ejemplos de uso

Ejemplo 1: Crear la máscara de una imagen.

```
I = imread('glass.png');  
imshow(I)  
  
BW = imextendedmax(I,80);  
figure, imshow(BW)
```

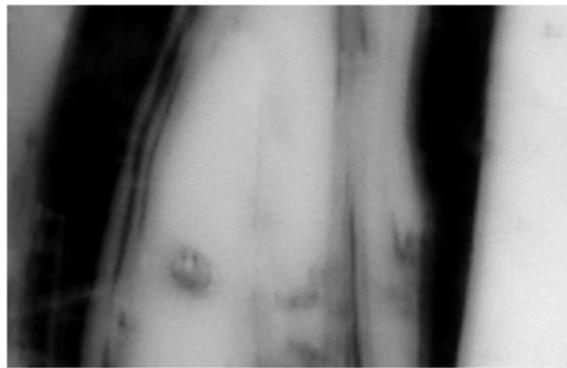


Figura f.263 - Imagen original (I) sin procesar



Figura f.264 - Imagen procesada (BW) con
imextendedmax

Ejemplo 2: Identificar los núcleos (zonas más claras) de las células de la imagen utilizando la Transformada extendida máxima comparando los diferentes resultados obtenidos al variar el valor de umbral máximo.

```
I = imread('celulas.png');  
  
% Sacamos los máximos regionales sin umbral.  
  
rmax = imregionalmax (I);  
imshow(I)  
  
% Sacamos los máximos regionales con umbral.  
  
I40 = imextendedmax(I, 40);  
I60 = imextendedmax(I, 60);  
  
% Superponemos los resultados en verde sobre la imagen original.  
  
o1 = imoverlay(I, rmax, [.3 1 .3]);  
o2 = imoverlay(I, I40, [.3 1 .3]);  
o3 = imoverlay(I, I60, [.3 1 .3]);
```

```
figure, imshow(o1)
figure, imshow(o2)
figure, imshow(o3)
```

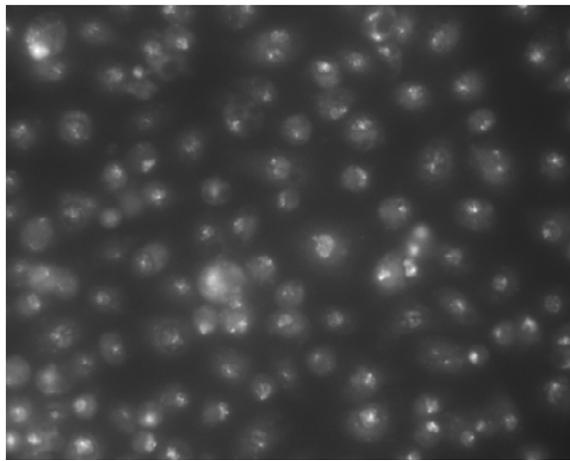


Figura f.265 - Imagen original (I) sin procesar

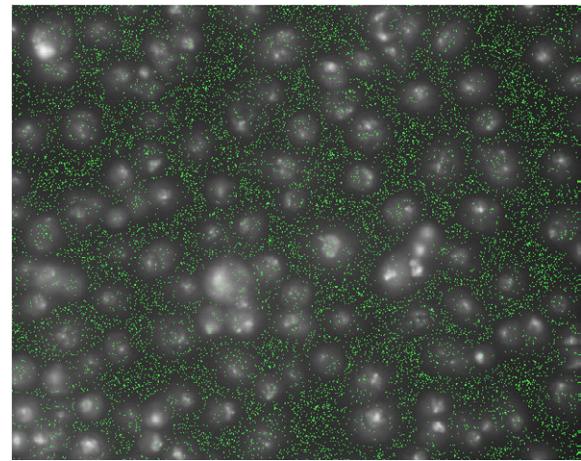


Figura f.266 - Imagen ($o1$) con los máximos regionales de I superpuestos

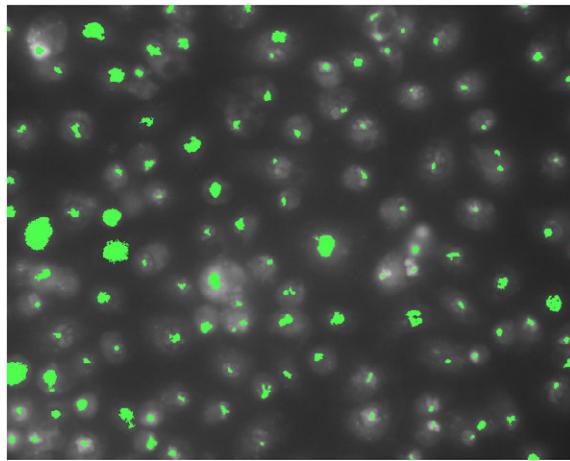


Figura f.267 - Imagen ($o2$) con los máximos regionales de I con umbral 40 superpuestos

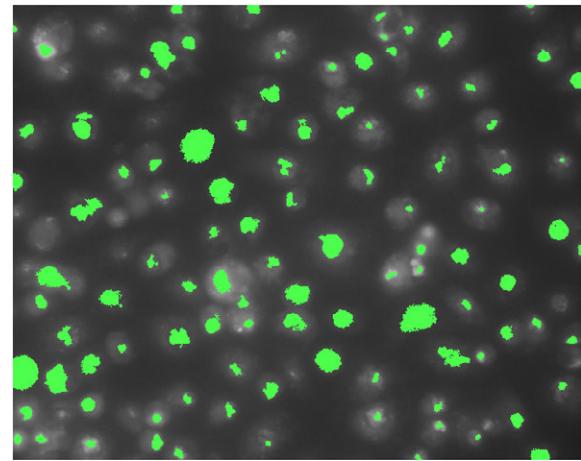


Figura f.268 - Imagen ($o3$) con los máximos regionales de I con umbral 60 superpuestos

Funciones relacionadas

conndef, imextendedmin, imhmax, imreconstruct, imregionalmax

imextendedmin

Transformada extendida mínima

Sintaxis

```
BW = imextendedmin(I,h)
BW = imextendedmin(I,h,conn)
```

Descripción

`BW = imextendedmin(I,h)` calcula la transformada extendida mínima de la imagen `I`. `h` es un escalar no negativo que especifica el umbral o diferencia que debe existir entre los píxeles mínimos regionales y los píxeles vecinos. Es una forma de identificar grupos de píxeles que tienen un valor de intensidad significativamente más bajo que el de sus vecinos. Para más información consultar la página de la función complementaria `imextendedmax`.

`imextendedmin` utiliza por defecto vecindades de 8 píxeles para imágenes en 2-D y vecindad de 26 píxeles para imágenes en 3-D. Para dimensiones mayores, `imextendedmin` utiliza la conectividad dada por `conndef(ndims(I), 'maximal')`.

`BW = imextendedmin(I,h,conn)` calcula la transformada extendida mínima donde `conn` especifica la conectividad y puede tener cualquiera de los siguientes valores escalares (siempre simétrica respecto su elemento central):

Valor	Significado
Conejividades Bidimensionales	
4	Vecindad de 4 píxeles: Un par de píxeles colindantes forman parte del mismo objeto solo si están activos y conectados en vertical u horizontal.
8	Vecindad de 8 píxeles: Un par de píxeles colindantes forman parte del mismo objeto solo si están activos. (Por defecto)
Conejividades Tridimensionales	
6	Vecindad de 6 píxeles.

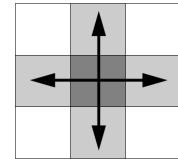


Figura f.32 - vecindad 4

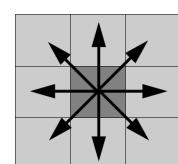


Figura f.33 - vecindad 8

18	Vecindad de 18 píxeles.
26	Vecindad de 26 píxeles. (Por defecto)

Nota: La conectividad se puede definir para cualquier dimensión de forma más general utilizando para `conn` una matriz de 3-por-3-por-...-por-3 de ceros y unos. Los unos definen las localizaciones de los píxeles vecinos relativas al elemento central de `conn`. Por ejemplo, `conn=ones(3)` define una conectividad de 8 píxeles.

`I` puede ser de cualquier clase numérica no dispersa y de cualquier dimensión. `BW` tiene el mismo tamaño que `I` y es siempre de tipo lógico.

Ejemplos de uso

Ejemplo 1: Crear la máscara de una imagen.

```
I = imread('glass.png');
BW = imextendedmin(I,50);
imshow(I), figure, imshow(BW)
```

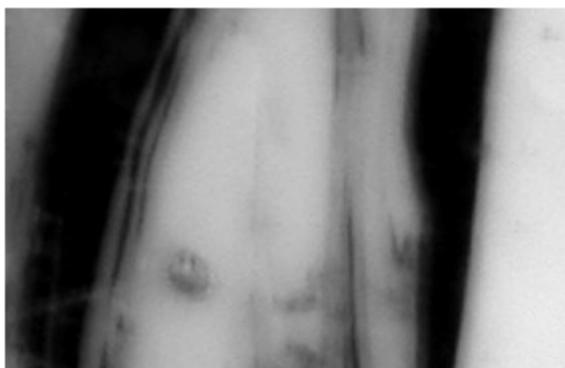


Figura f.263 - Imagen original (*I*) sin procesar



Figura f.269 - Imagen procesada (*BW*) con
imextendedmin

Funciones relacionadas

`conndef`, `imextendedmax`, `imhmin`, `imreconstruct`, `imregionalmin`

imfill

Relleno de regiones y huecos de una imagen

Sintaxis

```
BW2 = imfill(BW)
[BW2,locations] = imfill(BW)
BW2 = imfill(BW,locations)
BW2 = imfill(BW,'holes')
I2 = imfill(I)
BW2 = imfill(BW,locations,conn)
```

Descripción

`BW2 = imfill(BW)` muestra en pantalla la imagen binaria `BW` y permite definir la región a llenar interactivamente, especificando los puntos de la región con el ratón. Para poder usar esta herramienta interactiva `BW` debe ser una imagen en 2-D. Se debe pulsar las teclas de retroceso o borrado para eliminar el punto seleccionado. Pulsando la tecla Shift, haciendo clic derecho o doble clic, se selecciona el punto final y se empieza la operación de relleno. Al pulsar la tecla Intro se termina la selección sin añadir ningún punto más.

`[BW2,locations] = imfill(BW)` devuelve las localizaciones de los puntos seleccionados (región a llenar) de forma interactiva en la variable `locations`. Para utilizar esta sintaxis `BW` debe ser una imagen en 2-D .

`BW2 = imfill(BW,locations)` rellena los píxeles de fondo de la imagen binaria `BW` especificados en la variable `locations`. `locations` es un vector que puede contener el índice lineal de una serie de puntos que marcan la región a llenar o el índice lineal de un punto que marque dónde comenzará la operación de relleno.

`BW2 = imfill(BW,'holes')` rellena los huecos en la imagen binaria `BW`. En esta sintaxis un hueco es un conjunto de píxeles de fondo que no se pueden alcanzar llenando el fondo desde el borde de la imagen.

`I2 = imfill(I)` rellena los huecos en la imagen en escala de grises `I`. En esta sintaxis un hueco es una zona de píxeles oscuros rodeada de píxeles más claros.

`BW2 = imfill(BW,locations,conn)` rellena el área definida por la variable `locations`, donde `conn` especifica la conectividad y puede tener cualquiera de los siguientes valores escalares (siempre simétrica respecto su elemento central):

Valor	Significado
Conectividades Bidimensionales	
4	Vecindad de 4 píxeles: Un par de píxeles colindantes forman parte del mismo objeto solo si están activos y conectados en vertical u horizontal.
8	Vecindad de 8 píxeles: Un par de píxeles colindantes forman parte del mismo objeto solo si están activos. (Por defecto)
Conectividades Tridimensionales	
6	Vecindad de 6 píxeles.
18	Vecindad de 18 píxeles.
26	Vecindad de 26 píxeles. (Por defecto)
<p>Nota: La conectividad se puede definir para cualquier dimensión de forma más general utilizando para <code>conn</code> una matriz de 3-por-3-por...-por-3 de ceros y unos. Los unos definen las localizaciones de los píxeles vecinos relativas al elemento central de <code>conn</code>. Por ejemplo, <code>conn=ones(3)</code> define una conectividad de 8 píxeles.</p>	

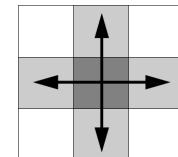


Figura f.32 - vecindad 4

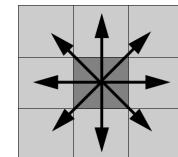


Figura f.33 - vecindad 8

`imfill` utiliza por defecto una vecindad de 4 píxeles de fondo para entradas en 2-D y vecindad de 6 píxeles de fondo para entradas en 3-D. Para dimensiones mayores, la conectividad del fondo por defecto viene determinada por `conndef(NUM_DIMS,'minimal')`. Se puede anular la conectividad por defecto con cualquiera de las siguientes sintaxis:

```
BW2 = imfill(BW,locations,conn)
BW2 = imfill(BW,conn,'holes')
I2 = imfill(I,conn)
```

Para anular la conectividad por defecto y además especificar las localizaciones de partida se debe utilizar la siguiente sintaxis:

```
BW2 = imfill(BW,0,conn)
```

La imagen de entrada puede tener cualquier dimensión, puede ser de clase numérica o lógica y debe ser real y no dispersa. La imagen de salida será de la misma clase que la de entrada.

Ejemplos de uso

Ejemplo 1: Rellenar los huecos de una imagen binaria primero partiendo de una localización determinada de uno de los píxeles hueco, luego aplicando un relleno especificando todos los píxeles hueco y finalmente llenando los huecos sin especificar localización.

```
BW1 = logical([1 0 0 0 0 0 0 0  
              1 1 1 1 1 0 0 0  
              1 0 0 0 1 0 1 0  
              1 0 0 0 1 1 1 0  
              1 1 1 1 0 1 1 1  
              1 0 0 1 1 0 1 0  
              1 0 0 0 1 0 1 0  
              1 0 0 0 1 1 1 0])  
  
% Rellenamos los huecos que se encuentran conectados con conectividad 8  
(en gris oscuro) especificando uno de esos píxeles hueco (en blanco y  
negrita) para que desde ahí rellene todos los huecos conectados.  
  
BW2 = imfill(BW1,[3 3],8)  
  
BW1 =  
      1 0 0 0 0 0 0 0  
      1 1 1 1 1 0 0 0  
      1 0 0 0 1 0 1 0  
      1 0 0 0 1 1 1 0  
      1 1 1 1 0 1 1 1  
      1 0 0 1 1 0 1 0  
      1 0 0 0 1 0 1 0  
      1 0 0 0 1 1 1 0  
  
BW2 =  
      1 0 0 0 0 0 0 0  
      1 1 1 1 1 0 0 0  
      1 1 1 1 1 0 0 0  
      1 1 1 1 1 1 1 0  
      1 1 1 1 1 1 1 1  
      1 0 0 1 1 1 1 1  
      1 0 0 0 1 1 1 0  
      1 0 0 0 1 1 1 0  
  
% Rellenamos los huecos que se encuentran conectados con conectividad 8  
especificando cada uno de ellos en la variable locations.  
  
locations = [18;19;20;26;27;28;37;46;54];  
  
BW2 = imfill(BW1,locations,8)  
  
BW2 =
```

```
1     0     0     0     0     0     0     0
1     1     1     1     1     0     0     0
1     1     1     1     1     0     1     0
1     1     1     1     1     1     1     0
1     1     1     1     1     1     1     1
1     1     1     1     1     1     1     1
1     0     0     1     1     1     1     0
1     0     0     0     1     1     1     0
1     0     0     0     0     1     1     0
```

% Rellenamos los huecos que se encuentran conectados con conectividad 8 sin especificar el punto de inicio de relleno.

% Primero considerando que un hueco es un conjunto de píxeles de fondo que no se pueden alcanzar rellenando el fondo desde el borde de la imagen. En este caso no habría huecos a llenar.

```
BW2 = imfill(BW1,8)
```

```
BW2 =
```

```
1     0     0     0     0     0     0     0
1     1     1     1     1     0     0     0
1     0     0     0     1     0     1     0
1     0     0     0     1     1     1     0
1     1     1     1     0     1     1     1
1     0     0     1     1     0     1     0
1     0     0     0     1     0     1     0
1     0     0     0     0     1     1     0
```

% Ahora considerando que un hueco es un conjunto de píxeles de fondo rodeados de píxeles blancos. En este caso sí habría huecos a llenar.

```
BW2 = imfill(BW1,8,'holes')
```

```
BW2 =
```

```
1     0     0     0     0     0     0     0
1     1     1     1     1     0     0     0
1     1     1     1     1     0     1     0
1     1     1     1     1     1     1     0
1     1     1     1     1     1     1     1
1     0     0     1     1     1     1     0
1     0     0     0     1     1     1     0
1     0     0     0     0     1     1     0
```

Ejemplo 2: Rellenar los huecos de una imagen binaria.

```
I = imread('circuloshueco.tif');
imshow(I)
```

```
BW = imfill(I,'holes');
figure, imshow(BW)
```

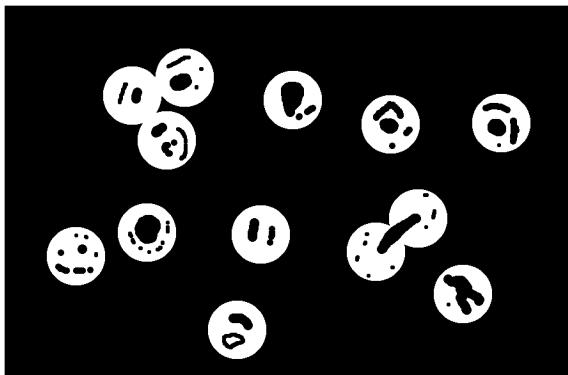


Figura f.270 - Imagen original (*I*) sin procesar

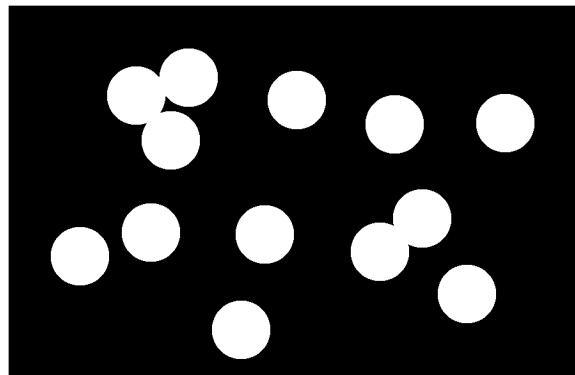


Figura f.48 - Imagen procesada (*BW*) con *imfill*

Funciones relacionadas

`bwselect`, `imreconstruct`, `roifill`

imfilter

Filtrado espacial lineal de imágenes multidimensionales

Sintaxis

```
B = imfilter(A, H)
B = imfilter(A, H, option1, option2, ...)
```

Descripción

`B = imfilter(A, H)` filtra el array multidimensional `A` con el filtro multidimensional `H`. El array `A` puede ser de tipo lógico o un array numérico no disperso de cualquier clase y dimensión. `B` será del mismo tamaño y clase que `A`, luego `imfilter` es más eficiente que otras operaciones de filtrado.

Cada elemento de la variable de salida `B` se calcula utilizando precisión doble. Si `A` es un array de enteros o un array lógico, se truncarán los elementos de salida que excedan el rango de los enteros y se redondearán los valores fraccionarios.

`B = imfilter(A, H, option1, option2, ...)` realiza un filtrado multidimensional de acuerdo a las opciones especificadas, que pueden tomar los siguientes valores:

Opciones de frontera (determinan cómo será el borde exterior de la imagen de salida)

Opción	Descripción
<code>x</code>	Se asume que los valores del array de entrada fuera de las fronteras exteriores (fuera de la imagen) tienen el valor <code>x</code> . Cuando no se especifica ninguna opción de frontera <code>imfilter</code> usa <code>x = 0</code> , por lo que puede aparecer un borde negro alrededor de la imagen de salida.
<code>'symmetric'</code>	Los valores de entrada que se encuentran fuera de las fronteras del array se calculan reflejando el array a lo largo del borde del array.
<code>'replicate'</code>	Se asume que los valores de entrada que se encuentran fuera de las fronteras del array equivalen al valor más cercano al borde del array.
<code>'circular'</code>	Los valores de entrada que se encuentran fuera de las fronteras del array se calculan asumiendo que el array de entrada es periódico.

Opciones de tamaño de salida

Opción	Descripción
'same'	El array de salida tendrá el mismo tamaño que el de entrada. (Por Defecto)
'full'	El array de salida es el resultado del filtrado completo, luego es mayor que el array de entrada.

Opciones de correlación y convolución

Opción	Descripción
'corr'	imfilter realiza un filtrado multidimensional utilizando correlación, la misma manera con la que la función filter2 realiza el filtrado. (Por Defecto)
'conv'	imfilter realiza un filtrado multidimensional utilizando convolución.

Ejemplos de uso

Ejemplo 1: Observar cómo se produce una imagen no deseada cuando la imagen de salida se convierte al tipo de imagen de entrada al filtrar una imagen.

```
I = imread('cuadros.tif'); imshow(I,[])
% Creamos un filtro lineal sencillo de 31 x 31 y filtramos con imfilter.
F = ones(31);
I2 = imfilter(I, F); figure, imshow(I2,[])
```

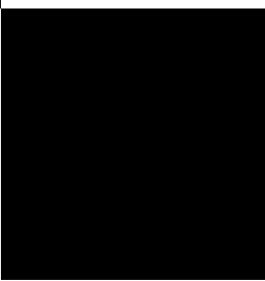
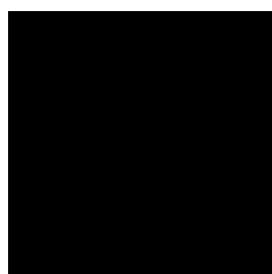


Figura f.271 - Imagen original (I) sin procesar

Figura f.272 - Imagen procesada (I2) con imfilter con los parámetros por defecto

Observamos como la imagen de salida no es la esperada, perdiendo datos. La razón es que la suma de los coeficientes de la máscara no está en el rango [0, 1] lo que ocasiona unos valores filtrados fuera del rango [0, 255] de las imágenes tipo `uint8`. Para evitar esto se deben normalizar los coeficientes de la máscara del filtro para que su suma esté en el rango [0, 1]. En el ejemplo anterior se haría dividiendo los coeficientes entre 31 x 31. Otra opción sería convertir la imagen de entrada a tipo `double`, aunque en este caso también podría ser necesario normalizar los valores posteriormente para guardar la imagen en un formato de imagen apropiado. En el siguiente ejemplo se utiliza esta solución.

Ejemplo 2: Comparar todas las opciones de frontera.

```
I = imread('cuadros.tif'); imshow(I,[])
F = ones(31);

% Escalamos los datos de entrada para que no se produzcan valores fuera
% del rango del tipo de imagen uint8 que es [0, 255].
I2 = im2double(I);

% Opción por defecto. Se rellena el borde de la imagen con ceros (negro),
% lo que crea un borde desenfocado entre la frontera y los cuadrados
% blancos.

I3 = imfilter(I2, F); figure, imshow(I3,[])

% Opción 'replicate' que no produce borde desenfocado entre la frontera y
% los cuadrados blancos.

I4 = imfilter(I2, F, 'replicate'); figure, imshow(I4,[])

% Opción 'symmetric'.

I5 = imfilter(I2, F, 'symmetric'); figure, imshow(I5,[])

% Opción 'circular' que vuelve a generar borde desenfocado entre la
% frontera y los cuadrados blancos debido al uso de la periodicidad, que
% crea partes negras asyacentes a las blancas.

I6 = imfilter(I2, F, 'circular'); figure, imshow(I6,[])
```

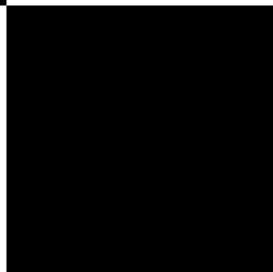
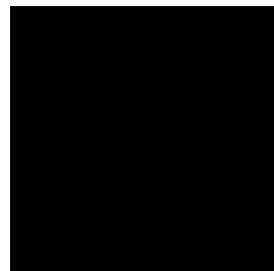


Figura f.271 - Imagen original (I) sin procesar



Figura f.273 - Imagen procesada (I3) con imfilter con los parámetros por defecto



Figura f.274 - Imagen procesada (I4) con imfilter con frontera tipo 'replicate'



Figura f.275 - Imagen procesada (I5) con imfilter con frontera tipo 'symmetric'



Figura f.276 - Imagen procesada (I6) con imfilter con frontera tipo 'circular'

Ejemplo 2: Aplicar un filtro de movimiento a una imagen con las opciones por defecto y después utilizar la opción de frontera 'replicate' para eliminar el borde negro.

```
RGB = imread('cherry.jpg');
imshow(RGB)

% Creamos un filtro que simula el movimiento lineal de una cámara.

h = fspecial('motion', 50, 45);

% Aplicamos el filtro con las opciones por defecto.

fRGB = imfilter(RGB, h);
figure, imshow(fRGB)

% Aplicamos el filtro utilizando una frontera de réplica para evitar el
% borde negro de la imagen de salida.

bfRGB = imfilter(RGB, h, 'replicate');
figure, imshow(bfRGB)
```



Figura f.277 - Imagen original (RGB) sin procesar



Figura f.278 - Imagen procesada (fRGB) con imfilter con los parámetros por defecto



Figura f.279 - Imagen procesada (bfRGB)
con imfilter con frontera tipo 'replicate'

Funciones relacionadas

conv2, convn, filter2, fspecial, ippl

imfreehand

Creación de una región a mano alzada

Sintaxis

```
h = imfreehand
h = imfreehand(hparent)
h = imfreehand(...,param1, val1,...)
```

Descripción

`h = imfreehand` inicia la herramienta interactiva de creación de una región a mano alzada en los ejes actuales de la figura. La función devuelve `h`, un puntero a un objeto `imfreehand`. La región a mano alzada se puede desplazar interactivamente utilizando el ratón y dispone de un menú contextual con acciones que controlan su apariencia y comportamiento.

`h = imfreehand(hparent)` inicia la herramienta interactiva de creación de una región a mano alzada en el objeto especificado por la variable `hparent`, típicamente unos ejes, pudiendo ser también cualquier objeto `hggroup`.

`h = imfreehand(...,param1, val1,...)` crea la región especificando parámetros y valores que controlan su comportamiento. La siguiente tabla muestra los parámetros disponibles. Los nombres de los parámetros se pueden abbreviar y no son sensibles a las mayúsculas:

Parámetro	Descripción
'Closed'	Valor escalar lógico que controla si la región es cerrada o no. Cuando es 1 (por defecto), <code>imfreehand</code> dibuja una línea recta para conectar los puntos finales de la línea y cerrar la región. Si su valor es 0, <code>imfreehand</code> deja abierta la región.
'PositionConstraintFcn'	Puntero a una función de restricción de posición que se llama cuando la región se desplaza interactivamente con el ratón. Usar este parámetro para controlar dónde puede moverse la región. Si se utiliza <code>imfreehand</code> en una imagen y no se especifica una función de restricción de posición, los usuarios podrán

	desplazar la elipse fuera de los límites de la imagen y perder la elipse. Cuando los ejes se crean con la función <code>plot</code> se expanden automáticamente para acomodar el movimiento de la elipse.
--	---

Cuando se llama a `imfreehand` utilizando la sintaxis interactiva, el puntero del ratón cambia a forma de cruz cuando se posiciona encima de la imagen. La región se crea haciendo clic con el ratón y desplazando el puntero. `imfreehand` dibuja por defecto una línea recta para unir los extremos de la región y cerrar la región. La siguiente tabla explica las acciones más comunes a realizar con la herramienta:

Acción	Operación
Mover la elipse.	Mover el puntero del ratón dentro de la región. El puntero cambiará a forma de cruz. Hacer clic y desplazar el ratón para mover la elipse.
Cambiar el color usado para dibujar la región.	Mover el puntero del ratón dentro de la región, hacer clic derecho y seleccionar Set Color (“Establecer color”) del menú contextual.
Leer la posición actual de la región.	Mover el puntero del ratón dentro de la región. Hacer clic derecho y seleccionar Copy Position (“Copiar posición”) del menú contextual. <code>imfreehand</code> copia al portapapeles un array de coordenadas de tamaño n-por-2.

Cada objeto `imfreehand` soporta una serie de métodos que pueden verse mediante el comando `methods imfreehand` y que se listan a continuación:

`addNewPositionCallback`

Añade una función a la lista de funciones de llamada o ‘callback’ de nueva posición (se llamará cada vez que se cambie la posición del objeto ROI (Region-of-interest) con la posición como argumento).

`createMask`

Crea una máscara dentro de la imagen.

`delete`

Elimina el objeto ROI.

`getColor`

Lee el color utilizado para dibujar el objeto ROI.

`getPosition`

Lee la posición actual de la elipse.

`getPositionConstraintFcn`

Devuelve el puntero a la función de restricción de posición actual.

`removeNewPositionCallback`

Elimina la función de llamada o ‘callback’ de nueva posición del objeto ROI.

`resume`

Continúa la ejecución de la línea de comandos.

`setClosed(h,TF)`

Especifica la geometría de la región. TF es un escalar lógico que marca si la región será cerrada (valor 1) o abierta (valor 0).

`setColor`

Especifica el color utilizado para dibujar el objeto ROI.

`setConstrainedPosition`

Especifica la nueva posición del objeto ROI.

`setPositionConstraintFcn`

Especifica la función de restricción de posición del objeto ROI.

`wait`

Bloquea la ejecución de la línea de comandos hasta que se finaliza el posicionamiento del objeto ROI.

Los comandos anteriores se ejecutan sobre h, el puntero al objeto `imfreehand`.

Ejemplos de uso

Ejemplo 1: Posicionar de forma interactiva una región a mano alzada y utilizar el método `wait` para bloquear la línea de comandos de MATLAB hasta que se haga doble clic en la región (se finalice la selección).

```
figure, imshow('pout.tif')
h = imfreehand;
position = wait(h);
```

Funciones relacionadas

`imellipse`, `imline`, `impoint`, `impoly`, `imrect`, `iptgetapi`,
`makeConstrainToRectFcn`

imgca

Obtención del puntero a los ejes de la imagen

Sintaxis

```
h = imgca  
h = imgca(hfig)
```

Descripción

`h = imgca` devuelve el puntero a los ejes actuales de la imagen para poder luego actuar sobre dichos ejes directamente utilizando otras funciones. Los ejes actuales pueden estar en una figura o una ventana de la herramienta imagen. Si no hay figuras que contengan ejes con imagen, `imgca` crea unos nuevos ejes.

`h = imgca(hfig)` devuelve el puntero a los ejes actuales de la imagen de la figura especificada (no hace falta que sea la figura actual).

Ejemplos de uso

Ejemplo 1: Hacer desaparecer los ejes de la figura actual y volver a hacerlos aparecer.

```
I = imread('circulos.tif');  
imshow(I)  
h = imgca;  
set(h, 'visible', 'off')  
pause  
set(h, 'visible', 'on')
```

Ejemplo 2: Calcular el centroide de cada círculo de la imagen binaria y superponerlos encima de cada círculo utilizando la función `imgca`:

```
I = imread('circulos.tif');  
imshow(I)  
  
% Obtenemos los centroides.  
  
s = regionprops(I, 'centroid');  
centroide = cat(1, s.Centroid);  
  
% Mostramos la imagen original en una nueva figura con los centroides  
superpuestos.  
  
imtool(I)  
hold(imgca, 'on')  
plot(imgca, centroide(:,1), centroide(:,2), 'r*')
```

```
hold(imgca,'off')
```

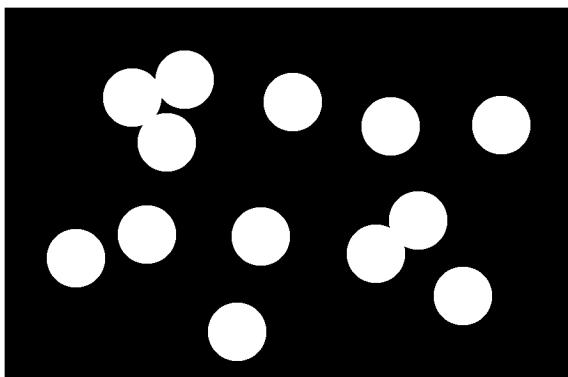


Figura f.48 - Imagen original (*I*) sin procesar

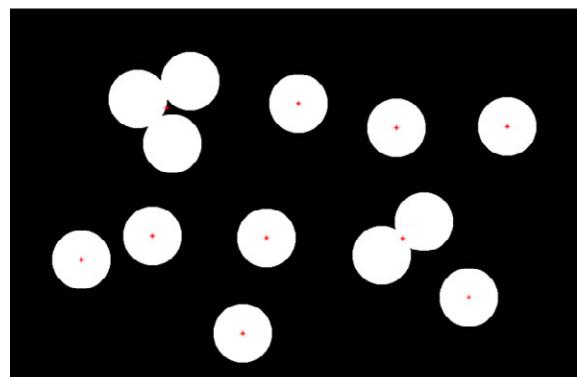


Figura f.280 - Imagen original con centroides

Funciones relacionadas

gca, gcf, imgcf, imhandles

imgcf

Obtención del puntero a la figura actual de la imagen

Sintaxis

```
hfig = imgcf
```

Descripción

hfig = imgcf devuelve el puntero a la figura actual que contiene la imagen para poder luego actuar sobre dicha figura directamente utilizando otras funciones. La figura puede ser una figura normal con al menos una imagen o una ventana de la herramienta imagen. Si ninguna de las figuras actuales abiertas contiene una imagen, imgcf crea una nueva figura.

Ejemplos de uso

Ejemplo 1: Abrir una imagen en una ventana de la herramienta imagen y cambiar su mapa de color utilizando la función imgcf.

```
imtool rice.png
cmap = copper(256);
set(imgcf,'Colormap',cmap)
```

Funciones relacionadas

`gca`, `gcf`, `imgca`, `imhandles`

imgetfile

Ventana para la lectura de la ruta de una imagen

Sintaxis

```
[filename, user_canceled] = imgetfile
```

Descripción

[filename, user_canceled] = imgetfile muestra la ventana de apertura de una imagen que permite seleccionar una imagen para leer su ruta completa en el espacio de trabajo. Se pueden seleccionar los tipos comunes de imagen soportados en MATLAB y especificados en imformats. Al seleccionar una imagen y hacer clic en el botón de Abrir, imgetfile devuelve la ruta completa a la imagen en la variable filename y la variable user_canceled con valor FALSE. Si el usuario hace clic en el botón de Cancelar o cierra la ventana, imgetfile devuelve una cadena vacía en la variable filename y la variable user_canceled con valor TRUE. Esta herramienta bloquea la línea de comandos de MATLAB mientras está en ejecución.

Funciones relacionadas

```
imformats, imtool, uigetfile
```

imhandles

Lectura de todos los punteros a imagen asociados con un puntero gráfico

Sintaxis

```
himage = imhandles(h)
```

Descripción

himage = imhandles(h) recibe un puntero gráfico h como entrada y devuelve todos los punteros a imagen asociados con él. h puede ser un array de punteros a figuras, ejes, imágenes u objetos uipanel. himage es un array de punteros a imágenes.

Ejemplos de uso

Ejemplo 1: Devolver el puntero a la imagen del eje actual.

```
figure, imshow('moon.tif')
himage = imhandles(gca)
```

Ejemplo 2: Mostrar dos imágenes en una misma figura y utilizar imhandles para leer los punteros a ambas imágenes.

```
subplot(1,2,1), imshow('autumn.tif')
subplot(1,2,2), imshow('glass.png')
himages = imhandles(gcf)
```

Funciones relacionadas

imgca, imgcf

imhist

Creación y visualización del histograma de una imagen

Introducción

El histograma es un gráfico que relaciona los niveles de intensidad de gris de una imagen y el número de píxeles que hay para cada nivel de intensidad. El histograma, además de caracterizar la imagen, se puede utilizar también para la segmentación de imágenes.

La función de la IPT para crear histogramas es `imhist`, pero para mostrar el histograma (los datos devueltos por `imhist`) existen también otras funciones que permiten tener más control sobre la representación, como son: `bar`, `stem` o `plot`. Se compararán todas estas funciones en un ejemplo posterior.

Sintaxis

```
imhist(I)
imhist(I, n)
imhist(X, map)
[counts,x] = imhist(...)
```

Descripción

`imhist(I)` muestra el histograma de la imagen `I` en escala de grises. El número de barras y los intervalos de intensidad utilizados en el histograma dependen del tipo de imagen. Por ejemplo si `I` es una imagen en escala de grises, `imhist` utiliza 256 barras e intervalos de intensidad por defecto y si `I` es una imagen binaria, `imhist` utiliza dos. Podemos normalizar el histograma mediante la expresión: `imhist(I)/numel(I)`.

`imhist(I, n)` muestra el histograma donde `n` especifica el número de barras e intervalos de intensidad utilizados en el histograma. Si `I` es una imagen binaria, `n` solo puede valer 2. Por ejemplo, si especificamos `n=2` para una imagen de clase `uint8` (rango [0, 255]) estaremos creando dos rangos de intensidad: de 0 a 127 y de 128 a 255, por lo que el histograma resultante devuelto por la función tendrá dos valores únicamente: el primero con un valor igual al número de píxeles en la imagen con valores en el intervalo [0, 127] y el segundo igual pero para el intervalo [128, 255].

`imhist(x, map)` muestra el histograma para la imagen indexada `x`. Este histograma muestra la distribución de los valores de los píxeles sobre una barra de color que usa el mapa de color especificado en la variable `map`. El mapa de color debe de tener una longitud de al menos el mayor índice en `x`. El histograma tiene una barra e intervalo para cada entrada en el mapa de color.

`[counts,x] = imhist(...)` devuelve los valores del histograma en la variable `counts` y las localizaciones de las barras e intervalos en `x` para su posterior uso, por ejemplo para visualizarlo con las funciones `bar`, `stem` o `plot` (p.ej. `stem(x,counts)`). Para imágenes indexadas, `imhist` devuelve los valores del histograma para cada entrada del mapa de color; la longitud de `counts` es la misma que la longitud del mapa de color.

Si la imagen de entrada es de intensidades puede ser de clase `uint8`, `uint16`, `int16`, `single`, `double` o `logical`. Si la imagen de entrada es indexada puede ser de clase `uint8`, `uint16`, `single`, `double` o `logical`.

Nota: Puede que el máximo valor en el eje-y se reduzca automáticamente. Para mostrar el rango completo del eje se debe de llamar a `imhist` con la siguiente sintaxis:

```
[counts,x] = imhist(...)  
stem(x,counts)
```

Ejemplos de uso

Ejemplo 1: Mostrar el histograma de una imagen `uint8` (intervalo [0, 255]) con la función `imhist`, primero con el número de barras e intervalos por defecto y luego reduciéndolos a 16 y luego a 8.

```
I = imread('cameraman.tif');  
  
imhist(I)  
figure, imhist(I, 16)  
figure, imhist(I, 8)
```



Figura f.179 - Imagen original (*I*)

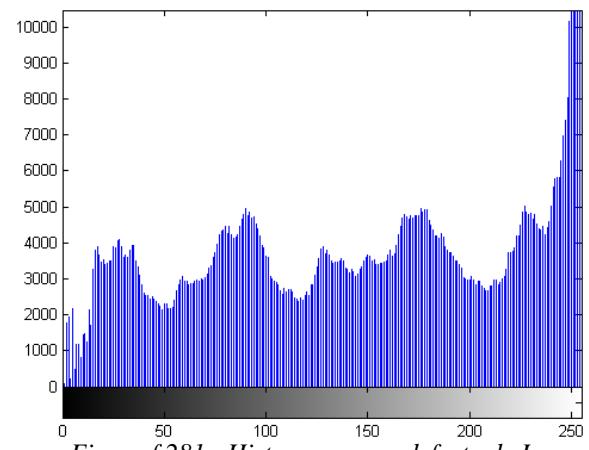


Figura f.281 - Histograma por defecto de *I*

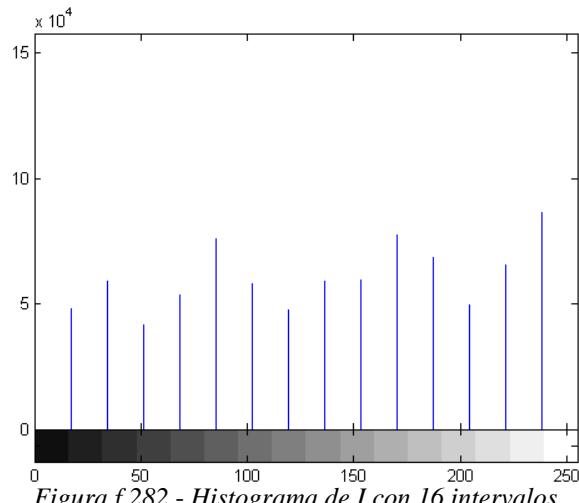


Figura f.282 - Histograma de *I* con 16 intervalos

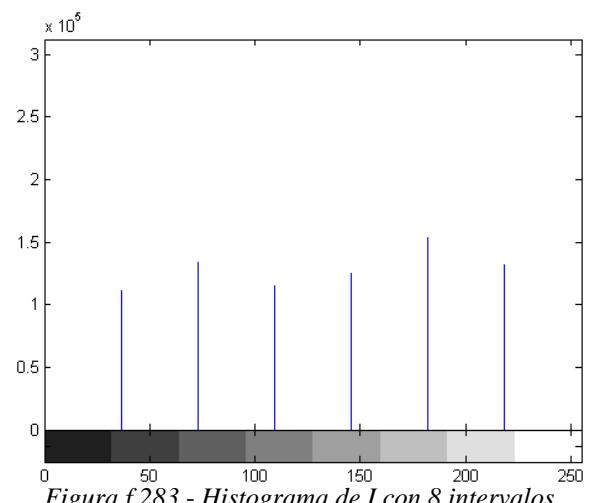


Figura f.283 - Histograma de *I* con 8 intervalos

Ejemplo 1: Comparar los histogramas visualizados mediante las funciones `imhist`, `bar`, `stem` y `plot`.

```

I = imread('cameraman.tif');

% Primero creamos, guardamos y mostramos el histograma con imhist.

[counts,x] = imhist(I);
imhist(I)

% Visualizamos el histograma con la función bar: bar(horz, v).
% v contiene los puntos que se representarán.
% horz es del mismo tamaño que v y tiene los incrementos horizontales.
% Se suele reducir la resolución horizontal para diferenciar las barras.
% En nuestro caso creamos grupos de 10 valores. Para mostrar todos los
valores se debe utilizar directamente la variable x en vez de horz.
    
```

```
h2 = counts(1:10:256);
horz = 1:10:256;
figure, bar(horz, h2)

% Le damos a la figura el mismo rango en el eje x que en la función
imhist y ampliamos el eje y para poder visualizar mejor los valores.

axis ([0 255 0 15000])

set(gca,'xtick',0:50:255) % Valores marcados en el eje x.
set(gca,'ytick',0:2000:15000) % Valores marcados en el eje y.

% Visualizamos el histograma con la función stem.

figure, stem(x,counts) % Mostramos todos los valores.
axis ([0 255 0 15000])
set(gca,'xtick',0:50:255)
set(gca,'ytick',0:2000:15000)

% Visualizamos el histograma con la función plot.

figure, plot(counts)
axis ([0 255 0 15000])
set(gca,'xtick',0:50:255)
set(gca,'ytick',0:2000:15000)
```

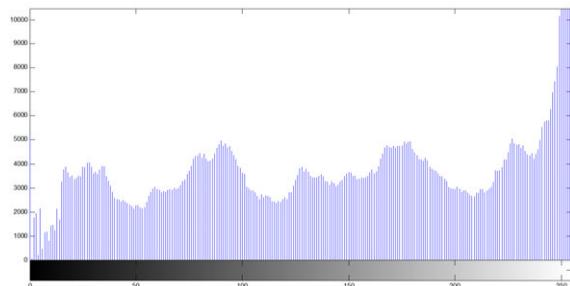


Figura f.284 - Histograma de I con hist

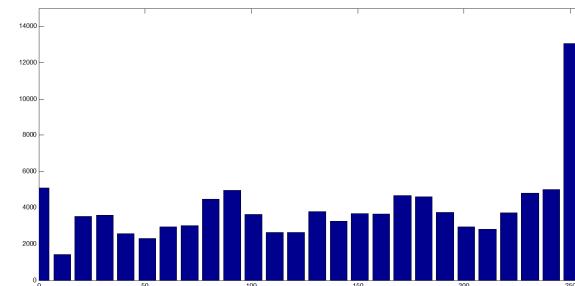


Figura f.285 - Histograma de I con bar

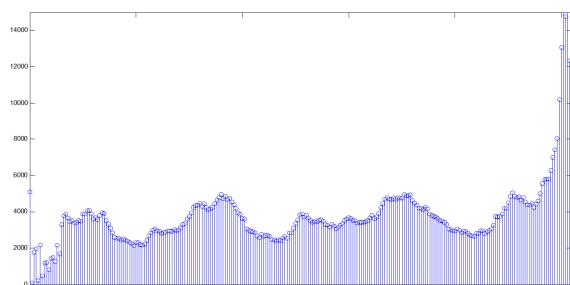


Figura f.286 - Histograma de I con stem

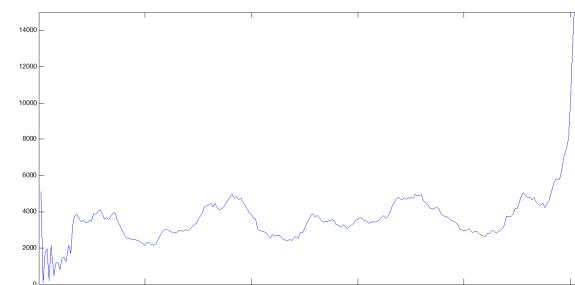


Figura f.287 - Histograma de I con plot

Funciones relacionadas

adaphisteq, histeq, hist

imhmax

Transformada H-máxima

Introducción

Un máximo (o mínimo) regional son un conjunto de píxeles conectados con un valor de intensidad constante y cuyos vecinos tienen un valor inferior (o mayor). Su altura (o profundidad) es la diferencia de valor. Por ejemplo, la siguiente imagen cuenta con dos máximos regionales principales (bloques de píxeles de valor 13 y 18) y varios máximos regionales inferiores de valor 11 con alturas de 3, 8 y 1 respectivamente.

A =	10	10	10	10	10	10	10	10	10	10
	10	13	13	13	10	10	11	10	11	10
	10	13	13	13	10	10	10	11	10	10
	10	13	13	13	10	10	11	10	11	10
	10	10	10	10	10	10	10	10	10	10
	10	11	10	10	10	18	18	18	10	10
	10	10	10	11	10	18	18	18	10	10
	10	10	11	10	10	18	18	18	10	10
	10	11	10	11	10	10	10	10	10	10
	10	10	10	10	10	10	11	10	10	10

Si se quiere limitar la altura de los máximos o la profundidad de los mínimos regionales a un cierto valor, se usan las funciones `imhmax` e `imhmin` (Transformada H-máxima y H-mínima), que eliminan cualquier máximo o mínimo regional en donde la diferencia entre el pixel regional y los píxeles vecinos sea menor (o mayor) que un cierto valor de umbral.

Sintaxis

```
I2 = imhmax(I,h)
I2 = imhmax(I,h,conn)
```

Descripción

`I2 = imhmax(I,h)` elimina cualquier máximo regional en la imagen de intensidades `I` en donde la diferencia entre el máximo regional y los píxeles vecinos (altura) sea menor que un cierto valor de umbral `h`. Se reducirán los máximos regionales existentes que tengan una diferencia mayor que la especificada en el umbral para que su diferencia coincida con dicho umbral. `h` es un escalar positivo que marca el umbral.

`imhmax` utiliza por defecto vecindades de 8 píxeles para imágenes en 2-D y vecindad de 26 píxeles para imágenes en 3-D. Para dimensiones mayores, `imhmax` utiliza la conectividad dada por `conndef(ndims(I), 'maximal')`.

`I2 = imhmax(I, h, conn)` calcula la transformada H-máxima donde `conn` especifica la conectividad y puede tener cualquiera de los siguientes valores escalares (siempre simétrica respecto su elemento central):

Valor	Significado
Conectividades Bidimensionales	
4	Vecindad de 4 píxeles: Un par de píxeles colindantes forman parte del mismo objeto solo si están activos y conectados en vertical u horizontal.
8	Vecindad de 8 píxeles: Un par de píxeles colindantes forman parte del mismo objeto solo si están activos. (Por defecto)
Conectividades Tridimensionales	
6	Vecindad de 6 píxeles.
18	Vecindad de 18 píxeles.
26	Vecindad de 26 píxeles. (Por defecto)
<p>Nota: La conectividad se puede definir para cualquier dimensión de forma más general utilizando para <code>conn</code> una matriz de 3-por-3-por-...-por-3 de ceros y unos. Los unos definen las localizaciones de los píxeles vecinos relativos al elemento central de <code>conn</code>. Por ejemplo, <code>conn=ones(3)</code> define una conectividad de 8 píxeles.</p>	

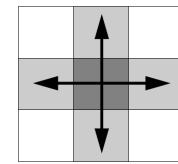


Figura f.32 - vecindad 4

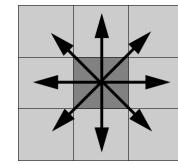


Figura f.33 - vecindad 8

`I` puede ser de cualquier clase numérica no dispersa y de cualquier dimensión. `I2` tiene el mismo tamaño y clase que `I`.

Ejemplos de uso

Ejemplo 1: Aplicar la Transformada H-máxima para limitar la altura de los máximos regionales y conseguir solo máximos regionales que difieran en 4 respecto sus píxeles vecinos.

```
a = zeros(10,10);
a(2:4,2:4) = 3; % máximo regional con diferencia 3 respecto sus vecinos.
a(6:8,6:8) = 8 % máximo regional con diferencia 8 respecto sus vecinos.

b = imhmax(a,4) % Transformada H-máxima con umbral 4.
```

```
a =
0 0 0 0 0 0 0 0 0 0
0 3 3 3 0 0 0 0 0 0
0 3 3 3 0 0 0 0 0 0
0 3 3 3 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 8 8 8 0 0
0 0 0 0 0 8 8 8 0 0
0 0 0 0 0 8 8 8 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0

b =
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 4 4 4 0 0
0 0 0 0 0 4 4 4 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

Funciones relacionadas

[conndef](#), [imextendedmax](#), [imhmin](#), [imreconstruct](#), [imregionalmax](#)

imhmin

Transformada H-mínima

Sintaxis

```
I2 = imhmin(I,h)
I2 = imhmin(I,h,conn)
```

Descripción

`I2 = imhmin(I,h)` elimina cualquier mínimo regional (píxeles conectados con un valor de intensidad constante y cuyos vecinos tienen un valor mayor) en la imagen de intensidades `I` en donde la diferencia entre los píxeles vecinos y el mínimo regional (profundidad) sea menor que un cierto valor de umbral `h`. Se aumentarán los mínimos regionales existentes que disten de los píxeles vecinos un valor mayor que el especificado en el umbral para que su diferencia coincida con dicho umbral. `h` es un escalar positivo que marca el umbral. Para más información consultar la página de la función complementaria `imhmax`.

`imhmin` utiliza por defecto vecindades de 8 píxeles para imágenes en 2-D y vecindad de 26 píxeles para imágenes en 3-D. Para dimensiones mayores, `imhmin` utiliza la conectividad dada por `conndef(ndims(I), 'maximal')`.

`I2 = imhmin(I,h,conn)` calcula la transformada H-mínima donde `conn` especifica la conectividad y puede tener cualquiera de los siguientes valores escalares (siempre simétrica respecto su elemento central):

Valor	Significado
Conectividades Bidimensionales	
4	Vecindad de 4 píxeles: Un par de píxeles colindantes forman parte del mismo objeto solo si están activos y conectados en vertical u horizontal.
8	Vecindad de 8 píxeles: Un par de píxeles colindantes forman parte del mismo objeto solo si están activos. (Por defecto)

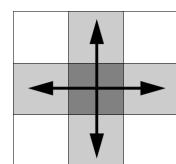


Figura f.32 - vecindad 4

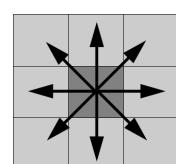


Figura f.33 - vecindad 8

Conejividades Tridimensionales

6	Vecindad de 6 píxeles.
18	Vecindad de 18 píxeles.
26	Vecindad de 26 píxeles. (Por defecto)

Nota: La conectividad se puede definir para cualquier dimensión de forma más general utilizando para `conn` una matriz de 3-por-3-por-...-por-3 de ceros y unos. Los unos definen las localizaciones de los píxeles vecinos relativas al elemento central de `conn`. Por ejemplo, `conn=ones(3)` define una conectividad de 8 píxeles.

`I` puede ser de cualquier clase numérica no dispersa y de cualquier dimensión. `I2` tiene el mismo tamaño y clase que `I`.

Ejemplos de uso

Ejemplo 1: Aplicar la Transformada H-mínima para conseguir solo mínimos regionales que difieran en 4 respecto sus píxeles vecinos.

```
a = 10*ones(10,10);
a(2:4,2:4) = 7; % mínimo regional con diferencia 3 respecto sus vecinos.
a(6:8,6:8) = 2 % mínimo regional con diferencia 8 respecto sus vecinos.

b = imhmin(a,4) % Transformada H-mínima con umbral 4.
```

```
a =
    10    10    10    10    10    10    10    10    10    10
    10     7     7     7    10    10    10    10    10    10
    10     7     7     7    10    10    10    10    10    10
    10     7     7     7    10    10    10    10    10    10
    10    10    10    10    10    10    10    10    10    10
    10    10    10    10    10     2     2     2    10    10
    10    10    10    10    10     2     2     2    10    10
    10    10    10    10    10     2     2     2    10    10
    10    10    10    10    10     2     2     2    10    10
    10    10    10    10    10    10    10    10    10    10

b =
    10    10    10    10    10    10    10    10    10    10
    10    10    10    10    10    10    10    10    10    10
    10    10    10    10    10    10    10    10    10    10
    10    10    10    10    10    10    10    10    10    10
    10    10    10    10    10    10    10    10    10    10
    10    10    10    10    10     6     6     6    10    10
    10    10    10    10    10     6     6     6    10    10
    10    10    10    10    10     6     6     6    10    10
    10    10    10    10    10    10    10    10    10    10
    10    10    10    10    10    10    10    10    10    10
```

Funciones relacionadas

`conndef`, `imextendedmin`, `imhmax`, `imreconstruct`, `imregionalmin`

imimposemin

Imposición de mínimo regional

Sintaxis

```
I2 = imimposemin(I,BW)
I2 = imimposemin(I,BW,conn)
```

Descripción

`I2 = imimposemin(I,BW)` modifica la imagen de intensidades `I` utilizando reconstrucción morfológica para que solo tenga un mínimo regional en la región positiva especificada por la máscara `BW`. `BW` es una imagen binaria del mismo tamaño que `I`.

Por defecto, `imhmin` utiliza vecindades de 8 píxeles para imágenes en 2-D y vecindad de 26 píxeles para imágenes en 3-D. Para dimensiones mayores, `imimposemin` utiliza la conectividad dada por `conndef(ndims(I), 'minimum')`.

`I2 = imimposemin(I,BW,conn)` especifica la conectividad en la variable `conn`, que puede tener cualquiera de los siguientes valores escalares (siempre simétrica respecto su elemento central):

Valor	Significado
Conectividades Bidimensionales	
4	Vecindad de 4 píxeles: Un par de píxeles colindantes forman parte del mismo objeto solo si están activos y conectados en vertical u horizontal.
8	Vecindad de 8 píxeles: Un par de píxeles colindantes forman parte del mismo objeto solo si están activos. (Por defecto)
Conectividades Tridimensionales	
6	Vecindad de 6 píxeles.
18	Vecindad de 18 píxeles.
26	Vecindad de 26 píxeles. (Por defecto)

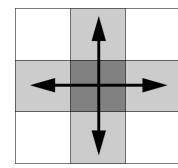


Figura f.32 - vecindad 4

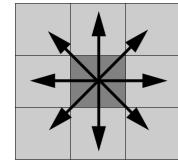


Figura f.33 - vecindad 8

Nota: La conectividad se puede definir para cualquier dimensión de forma más general utilizando para `conn` una matriz de 3-por-3-por-...-por-3 de ceros y unos. Los unos definen las localizaciones de los píxeles vecinos relativas al elemento central de `conn`. Por ejemplo, `conn=ones(3)` define una conectividad de 8 píxeles.

`I` puede ser de cualquier dimensión y clase numérica no dispersa. `BW` debe ser un array numérico no disperso del mismo tamaño que `I`. `I2` será del mismo tamaño y clase que `I`.

Ejemplos de uso

Ejemplo 1: Imponer un mínimo regional en una imagen dada.

```

I = 10*ones(10,10);
I(2:4,2:4) = 7; % mínimo regional.
I(6:8,6:8) = 2 % mínimo regional.

% Definimos dónde va a estar el mínimo regional utilizando zonas
positivas en una máscara binaria.

mask = zeros(10,10);
mask(7:9,2:4) = 1 % región donde habrá mínimo regional.

% Imponemos mínimo regional en la región positiva definida en mask.

J = imimposemin(I,mask);

% Comparamos los mínimos regionales antes y después de la operación.

MI = imregionalmin(I)
MJ = imregionalmin(I)

I =
  10   10   10   10   10   10   10   10   10   10
  10    7    7    7   10   10   10   10   10   10
  10    7    7    7   10   10   10   10   10   10
  10    7    7    7   10   10   10   10   10   10
  10   10   10   10   10   10   10   10   10   10
  10   10   10   10   10    2    2    2   10   10
  10   10   10   10   10    2    2    2   10   10
  10   10   10   10   10    2    2    2   10   10
  10   10   10   10   10   10   10   10   10   10
  10   10   10   10   10   10   10   10   10   10

mask =
  0    0    0    0    0    0    0    0    0    0
  0    0    0    0    0    0    0    0    0    0
  0    0    0    0    0    0    0    0    0    0
  0    0    0    0    0    0    0    0    0    0
  0    0    0    0    0    0    0    0    0    0
  0    0    0    0    0    0    0    0    0    0
  0    1    1    1    0    0    0    0    0    0
  0    1    1    1    0    0    0    0    0    0
  0    1    1    1    0    0    0    0    0    0
  0    0    0    0    0    0    0    0    0    0

```

MI =

```
0 0 0 0 0 0 0 0 0 0
0 1 1 1 0 0 0 0 0 0
0 1 1 1 0 0 0 0 0 0
0 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 0 0
0 0 0 0 0 1 1 1 0 0
0 0 0 0 0 1 1 1 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

MJ =

```
0 0 0 0 0 0 0 0 0 0
0 1 1 1 0 0 0 0 0 0
0 1 1 1 0 0 0 0 0 0
0 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 0 0
0 0 0 0 0 1 1 1 0 0
0 0 0 0 0 1 1 1 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

Ejemplo 2: Hacer que una imagen en escala de grises tenga solo un mínimo regional en un sitio específico.

```
I = imread('glass.tif');

% Definimos la zona donde estará el mínimo regional con una imagen
binaria del mismo tamaño que la original y con valor positivo solo en la
zona destinada al mínimo regional.

mask = false(size(I));
mask(65:70,65:70) = true;

% Mostramos la zona sobreimpresionada en la imagen original.

Z = I;
Z(mask) = 255;
figure, imshow(Z)

% Imponemos el mínimo regional en la imagen. Notar cómo todas las áreas
oscuras de la imagen original se han aclarado excepto el área definida
para el mínimo regional.

J = imimposemin(I,mask);
figure, imshow(J)

% Comparamos los mínimos regionales antes y después de la operación.

BW = imregionalmin(I);
figure, imshow(BW)
BW2 = imregionalmin(J);
figure, imshow(BW2)
```

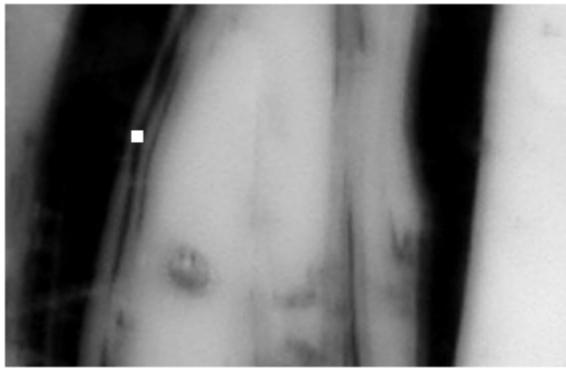


Figura f.263 - Imagen original (I) y máscara (mask) superpuesta

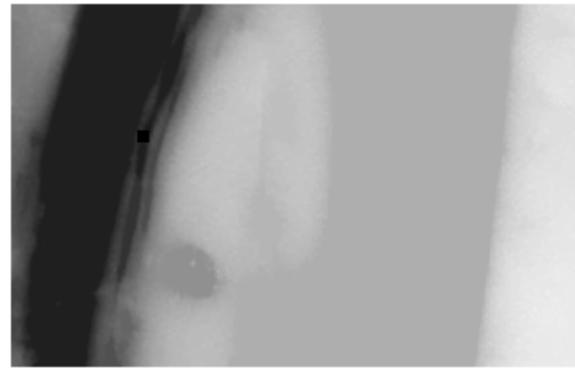


Figura f.288 - Imagen procesada (J)

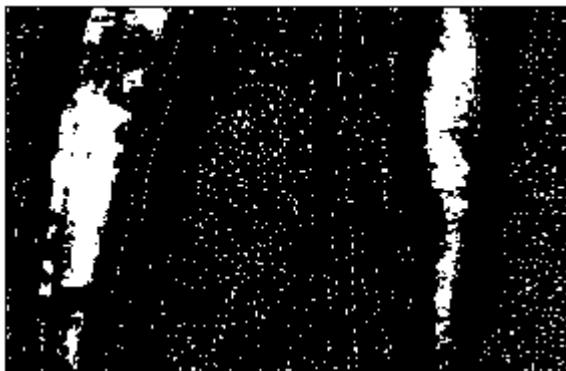


Figura f.289 - Mínimos regionales (BW) de imagen original (I)

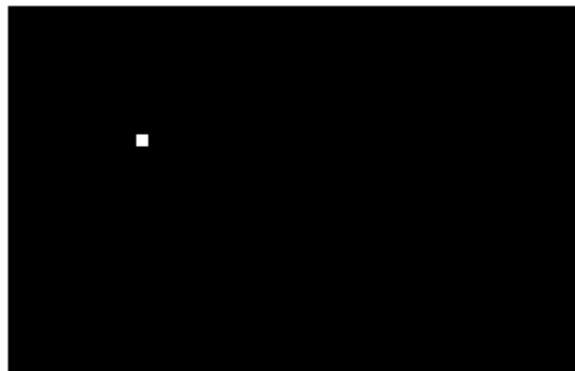


Figura f.290 - Mínimos regionales (BW2) de imagen procesada (J)

Funciones relacionadas

`conndef`, `imreconstruct`, `imregionalmin`