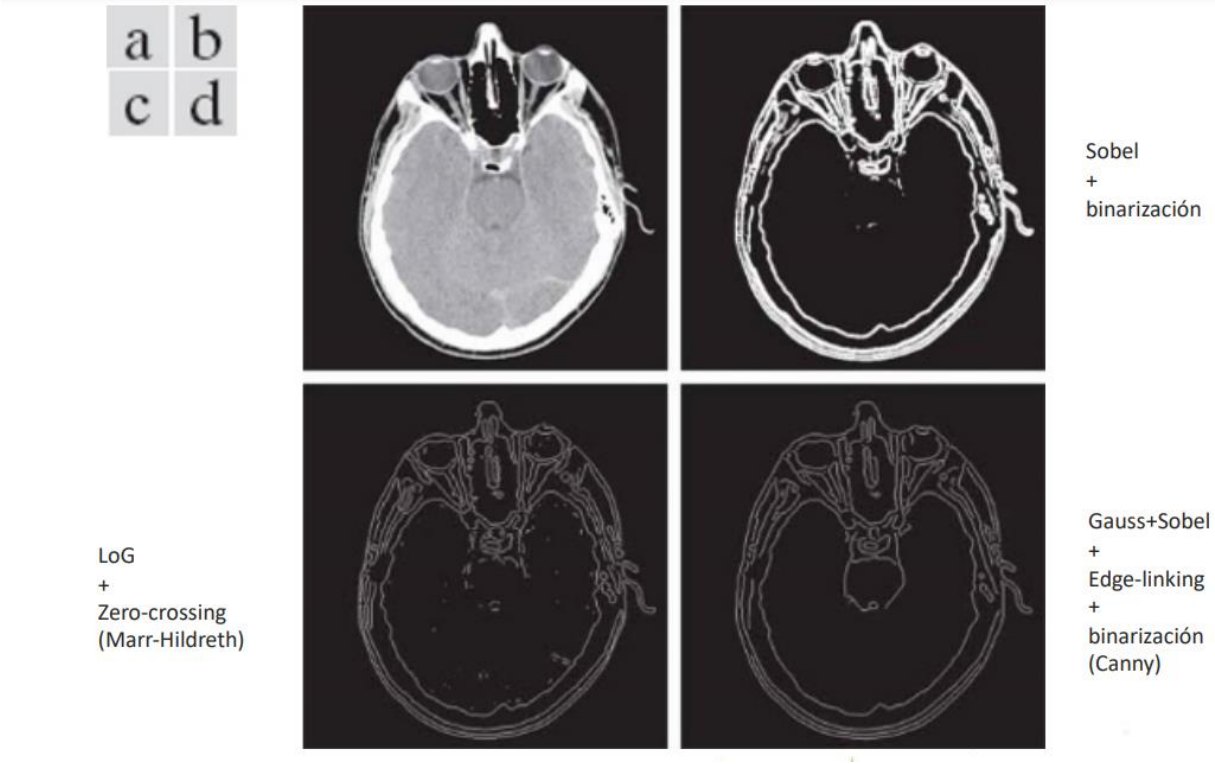
Universidad de Málaga

Materia: Imágenes biomédicas

Profesor: Ignacio Rodríguez Rodríguez y Enrique Nava Baro

Practica 4: Segmentación de imágenes II



Diego De Pablo

Málaga, noviembre de 2023

índice:

Introducción:

La segmentación es una tarea importante en el procesamiento de imágenes. Se utiliza para dividir una imagen en regiones que representan objetos o características de interés. La segmentación de imágenes tiene muchas aplicaciones en diferentes campos, como la medicina, la ingeniería, la ciencia y la seguridad.

En esta práctica, se utilizarán técnicas de segmentación de imágenes para delimitar los contornos de los objetos de interés en una imagen de resonancia magnética de un cráneo. Los objetos de interés en esta imagen son la masa cerebral y un tumor.

La práctica se dividirá en tres partes:

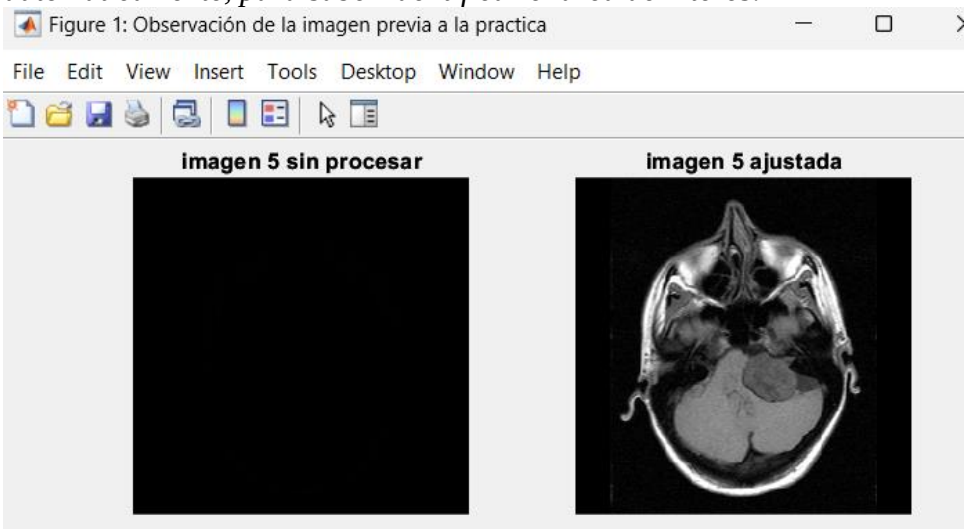
- En la primera parte, se utilizará un algoritmo de segmentación buscando destacar la masa cerebral y el tumor, entre los algoritmos que se pueden usar se encuentran:
    - Detección y extracción de bordes y contornos. Como el filtro de Sobel, los filtros de Canny, Marr-Hildreth o similares, basados en el uso del gradiente o la laplaciana
    - Crecimiento de regiones (regiongrow)
    - Split and merge
    - Contornos activos. como el método de Chan-Vese
    - Umbralización global o adaptativa
    - Uso de descriptores basados en texturas u otros parámetros estadísticos
    - Watershed
- La segunda parte consiste en buscar corregir los objetos, aplicar una erosión y una dilatación (una operación de cierre), para eliminar los objetos sobrantes, es decir aplicar post-procesado mediante el uso de filtros morfológicos para mejorar la calidad de la segmentación.
- En la tercera parte, se calcularán algunos descriptores de los objetos de interés, como el área, el perímetro y la compacidad.

El objetivo general de esta practica se podría decir que es lograr aplicar los conocimientos obtenidos durante las clases, en especial el conocimiento de las múltiples metodologías que se han visto a través del curso, destacando que no existe un método capaz de dar una solución perfecta para todo tipo de imágenes, algunos darán mejores resultados dependiendo el tipo de imagen que trabajemos.

También hay que recordar nuevamente que esta práctica esta realizada enfocada en una imagen de resonancia magnética del cráneo de un paciente, donde se toma como objeto de estudio la masa cerebral y tumor.

Contenido de la practica:

Antes de empezar quisiera recordar la imagen que toco y como se ve ajustada automáticamente, para saber identificar el área de interés:



Contenido de la práctica

1. Utilizar uno o varios (en cuyo caso la calificación será mejor) de los algoritmos de segmentación de imágenes (detección de bordes, crecimiento de regiones, contornos activos, etc.) para obtener uno o varios de los objetos de interés contenidos en la imagen utilizada. El profesor de la asignatura sugerirá opciones durante la sesión de práctica y proporcionará consejo sobre como conseguir los resultados, pero el trabajo y la elección debe ser individual. El resultado de estos algoritmos debe ser una imagen binaria.

Explicación:

Considero que antes de empezar aplicar cualquier método sería bastante beneficioso tener una idea clara de que hace cada método y para que tipos de imágenes son ideales ya que sabiendo la teoría podemos descartar (o quitarle prioridad) aquellos métodos que consuman mucho tiempo utilizar y probablemente no den una solución adecuada, así que recordaremos:

-Detección y extracción de bordes y contornos: es un método de segmentación que se basa en la identificación de los cambios abruptos de intensidad en una imagen. Los bordes son puntos o líneas que separan regiones con diferentes niveles de intensidad. La detección de bordes puede realizarse utilizando diferentes filtros, como el filtro de Sobel, el filtro de Laplaciano o el filtro de Canny. Una vez que se han detectado los bordes, se pueden extraer utilizando técnicas como la interpolación o el seguimiento de bordes.

Este método es adecuado para imágenes con bordes bien definidos, como imágenes de objetos industriales o de paisajes. En la practica 3 se vio que trabajar con este método puede traer soluciones rápidas, pero para obtener imágenes decentes debemos modificar bastante los métodos.

-Crecimiento de regiones: es un método de segmentación que se basa en la expansión de regiones a partir de una semilla. Las regiones se definen como conjuntos de píxeles que tienen valores similares. El crecimiento de regiones se inicia a partir de una o varias semillas. A continuación, se van expandiendo las regiones a partir de estas semillas, añadiendo los píxeles vecinos que tengan valores similares.

Este método es adecuado para imágenes con objetos de formas regulares, como imágenes de células o de tejidos. Promete buenos resultados para el caso de la imagen 5.

-Split and merge: es un método de segmentación que se basa en la división y fusión de regiones. Las regiones se dividen cuando se detectan cambios abruptos de intensidad entre ellas. Las regiones se fusionan cuando se encuentran regiones con valores similares.

El split and merge es un método relativamente complejo, pero puede ser muy efectivo para segmentar imágenes con objetos de formas irregulares. No exactamente nuestro caso, pero podría ser una solución valida.

-Contornos activos: Los contornos activos son un método de segmentación que se basa en la evolución de una curva a lo largo del borde de un objeto. La curva se mueve atraída por los bordes de la imagen y repelida por las regiones interiores. El método de Chan-Vese es particularmente útil para segmentar imágenes en regiones de primer plano y fondo. Comienza con una curva inicial (o contorno) que se especifica en una imagen, y luego la función activecontour evoluciona esta curva hacia los límites del objeto.

Los contornos activos son un método muy efectivo para segmentar objetos con bordes suaves o difusos. En el caso de la masa cerebral y el tumor tienen bordes relativamente bien definidos. Por lo tanto, los contornos activos pueden no ser la mejor opción para esta imagen.

-Umbralización global o adaptativa: son métodos de segmentación que se basa en la asignación de un valor de blanco o negro a cada píxel de una imagen, en función de su valor de intensidad. La *umbralización adaptativa* se realiza asignando un umbral diferente a cada píxel, en función de los valores de intensidad de los píxeles vecinos.

En este caso podríamos decir que la umbralización adaptativa funciona mejor para segmentar imágenes con una distribución de intensidades relativamente más complejas, en nuestro caso usar la umbralización adaptativa, es una opción vista en la practica 3.

-Uso de descriptores basados en texturas u otros parámetros estadísticos: es un método de segmentación que se basa en la identificación de las propiedades estadísticas de las regiones de una imagen. Los descriptores de textura pueden utilizarse para identificar regiones con texturas similares. Otros parámetros estadísticos, como la media, la desviación estándar o la correlación, pueden utilizarse para identificar regiones con propiedades estadísticas similares.

Este método es adecuado para imágenes con objetos que no tienen bordes bien definidos, como imágenes de textura o algunas imágenes médicas, pero no exactamente el caso de nuestra imagen cuyos bordes estan bien definidos. Este método podría ser útil sin embargo otros probablemente den mejores resultados con menor dificultad.

-Watershed: es un método de segmentación que se basa en la simulación del flujo de agua en una imagen. Las regiones se definen como las cuencas de drenaje que se forman cuando el agua fluye a través de los bordes de una imagen.

El watershed es un método muy efectivo para segmentar imágenes con bordes complejos.

*Codigo de Matlab de **CRECIMIENTO DE REGIONES**:*

```
%funcion dada por el profe basada en un umbral sobre el nivel de gris (González)
function [g, NR, SI, TI] = regiongrow(f, S, T)
f = double(f);
if numel(S) == 1
   SI = f == S;
   S1 = S;
else
   SI = bwmorph(S, 'shrink', Inf);
   J = find(SI);
   S1 = f(J); % Array of seed values.
end

TI = false(size(f));
for K = 1:length(S1)
   seedvalue = S1(K);
   S = abs(f - seedvalue) <= T;
   TI = TI | S;
end

[g, NR] = bwlabel(imreconstruct(SI, TI));
end
```

Resultado:

Explicando un poco la función regiongrow

La función regiongrow usada es prácticamente la dada por el profesor, obtenida del libro titulado: "Digital Image Processing Using MATLAB" (ver más información en la bibliografía especifica[1])

Entrada a la función

- **f:** La imagen de entrada a segmentar.
- **S:** La imagen de semillas, que puede ser un escalar que representa el valor de las semillas o una matriz binaria con 1s en las coordenadas de los puntos semilla y 0s en el resto de la imagen (solo se usará el caso de escalar en esta memoria).
- **T:** El umbral, que puede ser un escalar que representa un umbral global para toda la imagen o una matriz del mismo tamaño que la imagen f con valores de umbral para cada píxel (solo se usará el caso de escalar en esta memoria).

Salida de la función

- **g:** La imagen segmentada, donde cada región se etiqueta con un entero diferente.
- **NR**: El número de regiones identificadas en la segmentación.
- **SI:** La imagen de semillas final utilizada por el algoritmo.
- **TI:** La imagen que contiene los píxeles que satisfacen la prueba de umbral.

resumen paso a paso del código

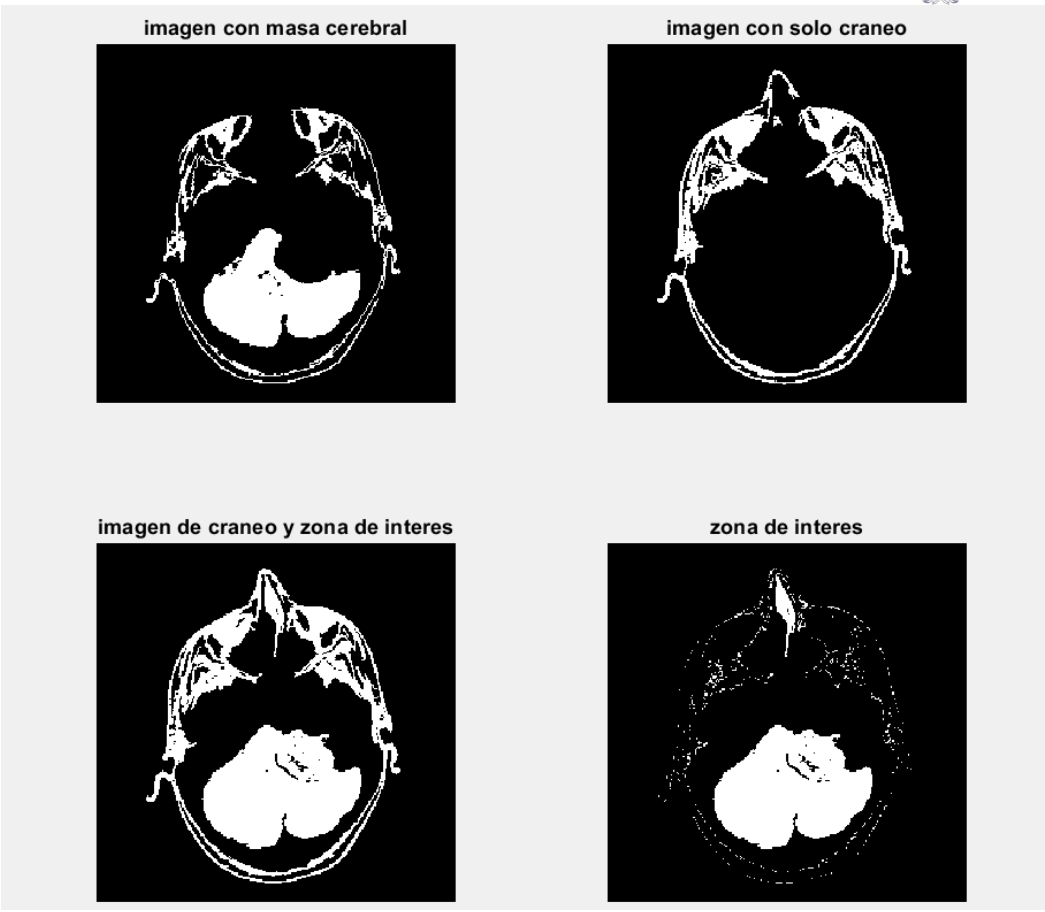
1. **Convertir la imagen de entrada a doble precisión:** Esto es necesario para el cálculo del umbral.

2. **Manejar la imagen de semillas escalar:** Este paso dependerá del valor dado a S, si es un escalar se crea una imagen binaria SI con 1s en los píxeles que corresponden al valor de la semilla en f. Si la imagen de semillas es una matriz, se elimina los puntos semilla duplicados y conectados para reducir el número de iteraciones del bucle en las siguientes secciones de código.
3. **Crear imagen de umbral:** Este paso crea una imagen de umbral TI que indica los píxeles dentro del umbral T del valor de semilla actual. Para ello, se itera a través de cada valor de semilla en S1. Para cada valor de semilla, se crea una imagen binaria S que indica los píxeles dentro del umbral T del valor de semilla actual.
4. **Realizar el crecimiento de regiones:** Para ello, se utiliza la función imreconstruct con la imagen de semillas SI y la imagen de umbral TI para reconstruir las regiones segmentadas. A continuación, se utiliza la función bwlabel para asignar etiquetas de enteros únicas a cada región.

Codigo de Matlab comparando distintos resultados con regiongrow*:*

```
%% Utilizar uno o varios de los algoritmos de segmentación de imágenes
%1.1 regiongrow usando S como escalar (regiongrow(f, S, T))
%imagen con 250 semillas y 75 umbral
[masa_cerebral, NR1, SI1, TI1] = regiongrow(imagen, 250, 75);
%imagen con 270 semillas y 120 umbral
[solo_craneo, NR2, SI2, TI2] = regiongrow(imagen, 270, 120);
%imagen con 250 semillas y 120 umbral
[craneo_y_cerebro, NR3, SI3, TI3] = regiongrow(imagen, 250, 120);

zona_interes = craneo_y_cerebro - solo_craneo;

figure('Name', 'Comparación de distintas semillas para regiongrow')
subplot(2,2,1);
imshow(masa_cerebral)
title('imagen con masa cerebral')
subplot(2,2,2);
imshow(solo_craneo)
title('imagen con solo craneo')
subplot(2,2,3);
imshow(craneo_y_cerebro)
title('imagen de craneo y zona de interes')
subplot(2,2,4);
imshow(zona_interes)
title('zona de interes')
```

Resultado:



Explicación:

Se usaron distintos valores de valor de semillas y del umbral, hasta que se encontraron imágenes destacables como puede ser la imagen superior izquierda la cual solo muestra la masa cerebral omitiendo el tumor, esto se debe a que el umbral usado fue muy pequeño (o también se podrían haber usado más semillas, ambos son términos que ayudan al ajuste de la umbralización, siempre y cuando no sé usen valores exagerados.

Arriba a la derecha podemos ver una aplicación muy curiosa donde muestra gran parte del cráneo, pero no muestra nada del objeto de interés, durante la practica 3 se dieron indicios de que los pixeles del cráneo ser tan parecidos a los pixeles del cerebro y del tumor sería útil aplicar una operación de resta, siendo esta imagen parte del cráneo que deseamos quitar de la imagen inferior izquierda que se muestra muy bien el área de la masa cerebral y el tumor.

Al restar la imagen inferior izquierda con la imagen superior derecha nos da de resultado la ultima imagen esta es un claro ejemplo de lo que deseamos obtener con la umbralización, nos deshicimos del fondo y de la mayor parte de aquellos objetos que no eran de interés como es el caso del cráneo, ahora solo queda aplicarle un filtro morfológico en el apartado 2 para conseguir una mejor umbralización.

A pesar de ser una buena umbralización considero que sería interesante comparar al menos con los otros métodos que teóricamente deberían mostrar también buenos resultados.

Codigo de Matlab de **Split and Merge***:*

```
function g = splitmerge(f, mindim, fun)
Q = 2^nextpow2(max(size(f)));
[M, N] = size(f);
f = padarray(f, [Q - M, Q - N], 'post');

S = qtdecomp(f, @split_test, mindim, fun);
Lmax = full(max(S(:)));
g = zeros(size(f));
MARKER = zeros(size(f));
% Begin the merging stage.
for K = 1:Lmax
   [vals, r, c] = qtgetblk(f, S, K);
   if ~isempty(vals)
      for I = 1:length(r)
         xlow = r(I); ylow = c(I);
         xhigh = xlow + K - 1; yhigh = ylow + K - 1;
         region = f(xlow:xhigh, ylow:yhigh);
         flag = feval(fun, region);
         if flag
            g(xlow:xhigh, ylow:yhigh) = 1;
            MARKER(xlow, ylow) = 1;
         end
      end
   end
end

% Finally, obtain each connected region and label it with a
% different integer value using function bwlabel.
g = bwlabel(imreconstruct(MARKER, g));

% Crop and exit
g = g(1:M, 1:N);
function v = split_test(B, mindim, fun)
k = size(B, 3);
v(1:k) = false;
for I = 1:k
   quadregion = B(:, :, I);
   if size(quadregion, 1) <= mindim
      v(I) = false;
      continue
   end
   flag = feval(fun, quadregion);
   if flag
      v(I) = true;
   end
end
end
%1.2.3 mejor resultado
function flag = predicate(region)
sd = std2(region);
m = mean2(region);
k=2.4;
M=150;
flag = abs(M-m)<k*sd
end
```

Explicación:

La función **splitmerge** realiza la segmentación de una imagen utilizando un algoritmo de división y fusión. La función toma tres argumentos de entrada:

- **f:** La imagen a segmentar.
- **mindim:** La dimensión mínima de las regiones del quadtree (subimágenes) permitidas. Si es necesario, el programa rellena la imagen de entrada con ceros hasta el siguiente tamaño cuadrado que es una potencia entera de 2. Esto garantiza que el algoritmo utilizado en la descomposición del quadtree podrá dividir la imagen en bloques de tamaño 1 por 1. El resultado se recorta al tamaño original de la imagen de entrada.
- **fun:** La función predicado. Esta función debe devolver TRUE si los píxeles en la región satisfacen el predicado definido por el código en la función; de lo contrario, el valor de FLAG debe ser FALSE.

resumen paso a paso del código

1. Rellena la imagen con ceros para garantizar que la función qtdecomp dividirá las regiones hasta un tamaño de 1 por 1.
2. Realiza la división primero. La función qtdecomp se utiliza para dividir la imagen en un quadtree.
3. Ahora fusiona mirando cada cuadrante y estableciendo todos sus elementos en 1 si el bloque satisface el predicado.
4. Finalmente, obtiene cada región conectada y la etiqueta con un valor entero diferente utilizando la función bwlabel.
5. Recorta y sale.

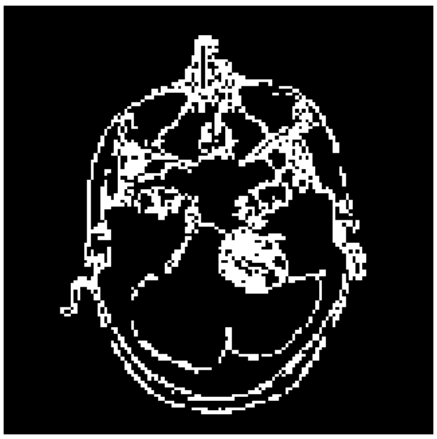
La función **split_test** es una función auxiliar de la función splitmerge. Determina si las regiones del quadtree se dividen. La función devuelve true para los bloques que deben dividirse y false para los que no deben dividirse. La función split_test realiza los siguientes pasos:

1. Comprueba si el tamaño del bloque es menor o igual que mindim. Si lo es, el bloque no se divide, por lo que el elemento correspondiente de v se establece en FALSE.
2. Comprueba si el bloque satisface el predicado. Si lo hace, el bloque se divide, por lo que el elemento correspondiente de v se establece en TRUE.

Codigo de Matlab:

```
%% 1.2 splitmerge v = split_test(B, mindim, fun)
%predicate(imagen)
splitmerge_final = splitmerge(imagen, 2, @predicate);
imshow(splitmerge_final)
```

Resultado:



Explicación:
La imagen se ve pixelada por la forma en la que trabaja el algoritmo, al usar mindin que es la dimensión mínima de las regiones del quadtree, es decir, controla el tamaño mínimo de las regiones que se etiquetarán como separadas. Haciendo que se logre detallar pero al tener un tamaño de estas regiones termina dando

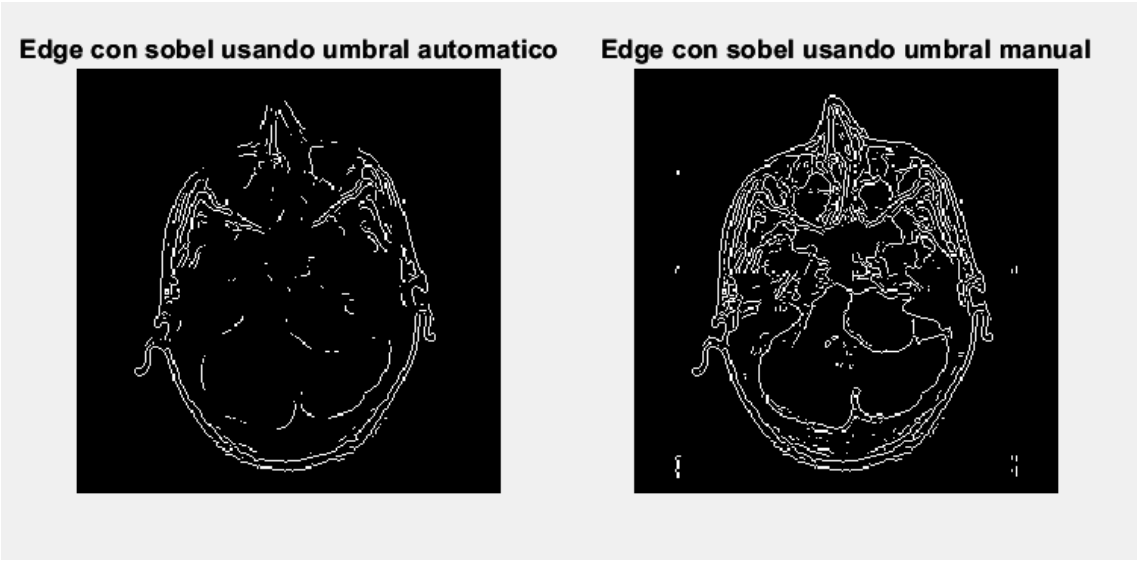
Sin embargo, hay algunas situaciones en las que aumentar el mindim puede ser beneficioso. Por ejemplo, si estás interesado en segmentar grandes regiones de la imagen, un valor más alto de mindim puede ayudarte a lograr mejores marcajes a los bordes pero pueden presentarse perdidas de información, un claro ejemplo es la siguiente imagen cuando se usa mindin = 4.

En última instancia, la decisión de aumentar o no el mindim depende de tus objetivos específicos. Si deseas obtener una imagen más detallada, es mejor mantener el mindim bajo. Si deseas segmentar grandes regiones de la imagen, puedes aumentar el mindim.

Utilización de algoritmos de detección y extracción de bordes y contornos de la imagen
Codigo de Matlab del Filtro sobel:

```
%% 1.3 Utilización de algoritmos de detección y extracción de bordes y contornos de la imagen
%1 Filtro sobel
[imagen_binaria,thresh]= edge(imagen, 'sobel'); %usa sobel por defecto
thresh; %ver umbral automatico

imagen_manual= edge(imagen, 'sobel',0.00035);

figure('Name', 'Visualización para función edge')
subplot(1,2,1);
imshow(imagen_binaria)
title('Edge con sobel usando umbral automatico')
subplot(1,2,2);
imshow(imagen_manual)
title('Edge con sobel usando umbral manual')
%codigo del libro:
%g = sqrt(imfilter(imagen,fs,'replicate').^2 + imfilter(imagen,fs','replicate').^2);
```

Resultado:



Explicación:
El filtro Sobel es un operador de detección de bordes que se utiliza en el procesamiento de imágenes. Calcula la aproximación del gradiente de la intensidad de una imagen, lo que puede ser útil para detectar cambios en la intensidad, o "bordes", en una imagen. El filtro Sobel es ideal para imágenes donde los bordes son importantes características que se deben resaltar. Sin embargo, es sensible al ruido en la imagen. Esto significa que, si la imagen tiene mucho ruido, los resultados pueden no ser óptimos ya que el ruido puede ser interpretado como bordes.

La función edge en MATLAB es una función de detección de bordes que se utiliza para identificar los puntos en una imagen donde la intensidad de los píxeles cambia de manera abrupta. La función edge puede utilizar varios métodos de detección de bordes, incluyendo el operador Sobel, que es el método predeterminado.

La función devuelve una imagen binaria que contiene 1s donde la función encuentra bordes en la imagen y 0s en otros lugares. Esto estos bordes se determinan según un umbral que puede ser generado automáticamente o dado manualmente por el usuario (en el código se ven ambos casos), donde se recomienda siempre intentar encontrar el valor óptimo de manera manual.

Se podría decir que es una forma más rápida de aplicar los filtros de detección de bordes. la principal diferencia entre usar edge y imfilter con un operador Sobel es que edge realiza algunas operaciones adicionales además de aplicar el operador Sobel. Estas operaciones adicionales pueden incluir el adelgazamiento de los bordes detectados y la eliminación de los bordes débiles.

En cuanto a los parámetros que puedes pasar a la función edge, aquí están algunos de los más comunes:

- **I**: la imagen
- **method**: Este parámetro especifica el método de detección de bordes que quieres usar. Puede ser 'sobel', 'prewitt', 'roberts', 'log' (Laplaciano de una Gaussiana), 'canny', 'approxcanny' and zero cross

- **thresh**: Este parámetro especifica el valor de umbral que se utiliza para determinar qué píxeles en la imagen constituyen un borde.
- **direction**: Este parámetro especifica la orientación de los bordes a detectar.
- **sigma:** Este parámetro especifica la desviación estándar del filtro, y sólo es válido cuando method es 'log' o 'canny'.

La función edge puede devolver varios valores:

- **BW:** Una imagen binaria que contiene 1s donde la función encuentra bordes en la imagen y 0s en otros lugares.
- **threshout:** El valor de umbral utilizado por la función edge.
- **Gx, Gy:** Los gradientes direccionales.

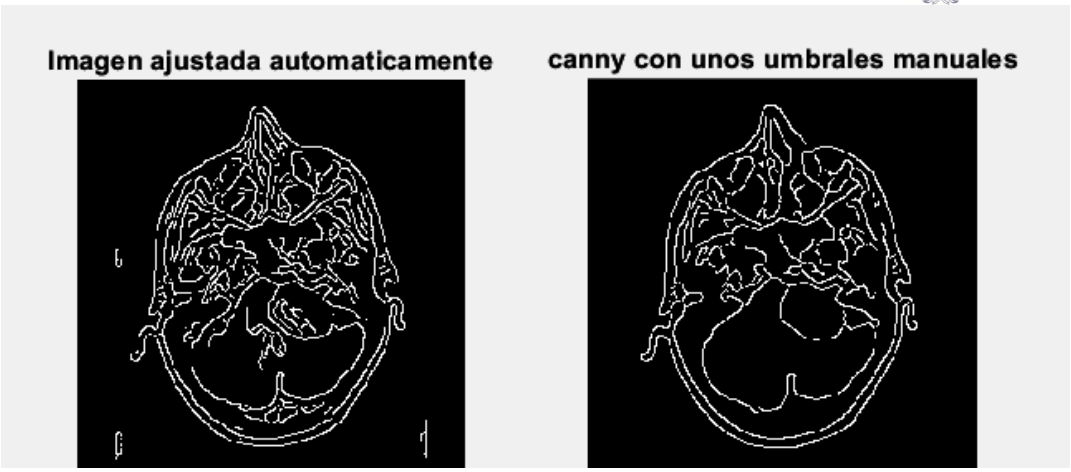
Cabe acotar que el umbral manual deja ver todo el borde entre la masa cerebral y el tumor, cosa que se pierde al dejar este valor automatizado, ya que cierta parte del borde no logra superar el umbral que se establece automáticamente de 8.7621e-04.

Esta es una segmentación donde se separa el fondo de los bordes de la imagen, lo cual no nos da exactamente la umbralizacion del objeto de interés, se podría hacer otro procesado para intentar separar la masa cerebral del craneo pero considero que teniendo tan buen resultado con el regiongrow estar más tiempo con el sobel no cunde para esta imagen, aunque en el apartado 2 se verá si con un postprocesado de filtros morfológicos podamos conseguir lo deseado

Codigo de Matlab de CANNY:

```
%% 1.3.2 (filtro canny)

[imagen_automatica,thresh]= edge(imagen, 'canny');
thresh; %ver umbral automatico
%imshow(imagen_automatica)

%canny manual
canny3= edge(imagen, 'canny', [0 0.1800],sqrt(3));


figure('Name', 'Visualización canny')
subplot(1,2,1);
imshow(imagen_automatica)
title('Imagen ajustada automaticamente')
subplot(1,2,2);
imshow(canny3)
title('canny con unos umbrales manuales')
```

Resultado:



Explicación:

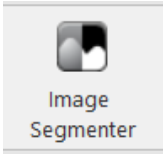
El método de Canny nombrado así por su creador, es un popular algoritmo de detección de bordes. Este algoritmo consta de múltiples etapas:

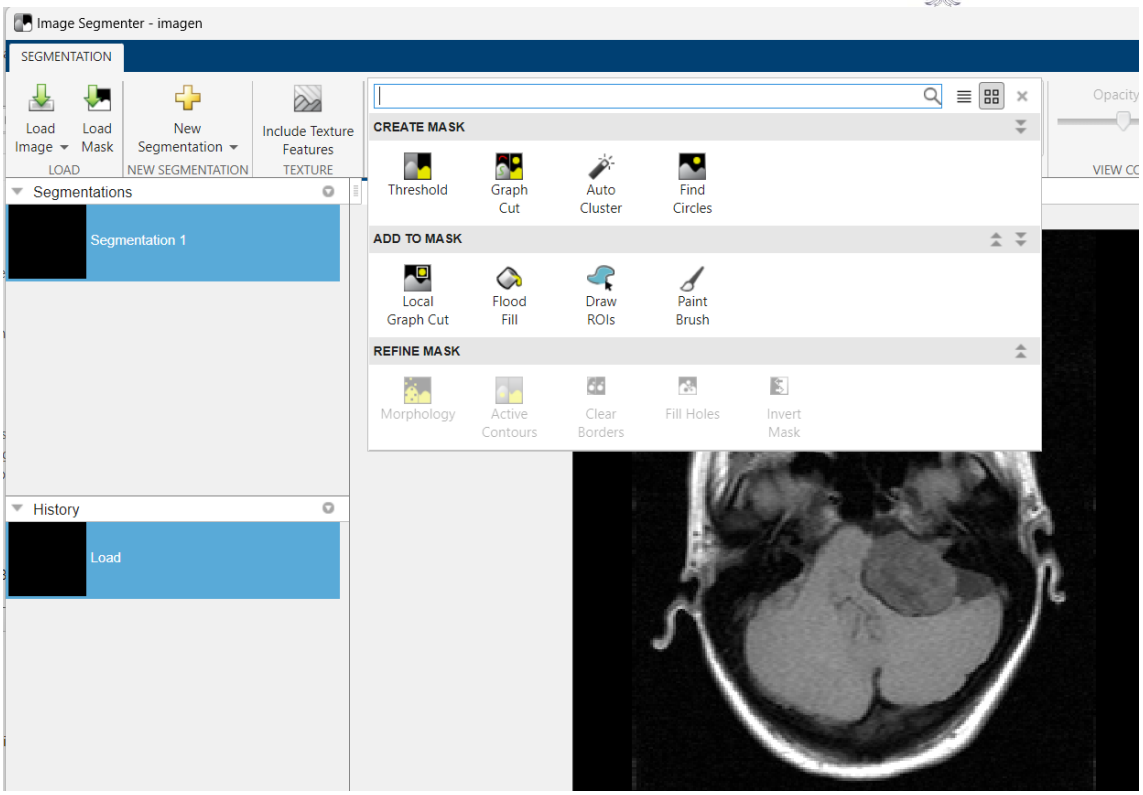
1. **Reducción de ruido**: Debido a que la detección de bordes es susceptible al ruido en la imagen, el primer paso es eliminarlo o reducirlo lo más que se pueda. Para esto, el algoritmo utiliza un filtro Gaussiano 5×5.
2. **Encontrando el gradiente de intensidad de la imagen**: Una vez que la imagen ha sido alisada con el filtro Gaussiano, se calcula el gradiente de esta.
3. **Supresión de falsos máximos**: Esta técnica es utilizada para afinar los bordes encontrados en el paso anterior.
4. **Umbral de histéresis**: En esta cuarta etapa se decide cuáles píxeles pertenecen realmente a bordes y cuáles no. Para ello, se deben fijar dos valores de umbral, minVal y maxVal.

El método de Canny difiere de otros en que utiliza dos umbrales diferentes (para detectar bordes intensos y débiles), e incluye los bordes débiles en el resultado solo si están conectados a bordes intensos

Considero que con estas umbralizaciones tenemos una idea clara de los distintos métodos mencionados teóricamente, por temas de comodidad se terminaran usando estos 4 que probablemente sean de los que muestren mejores resultados, pero se invita a seguir explorando con los demás métodos mencionados teóricamente.

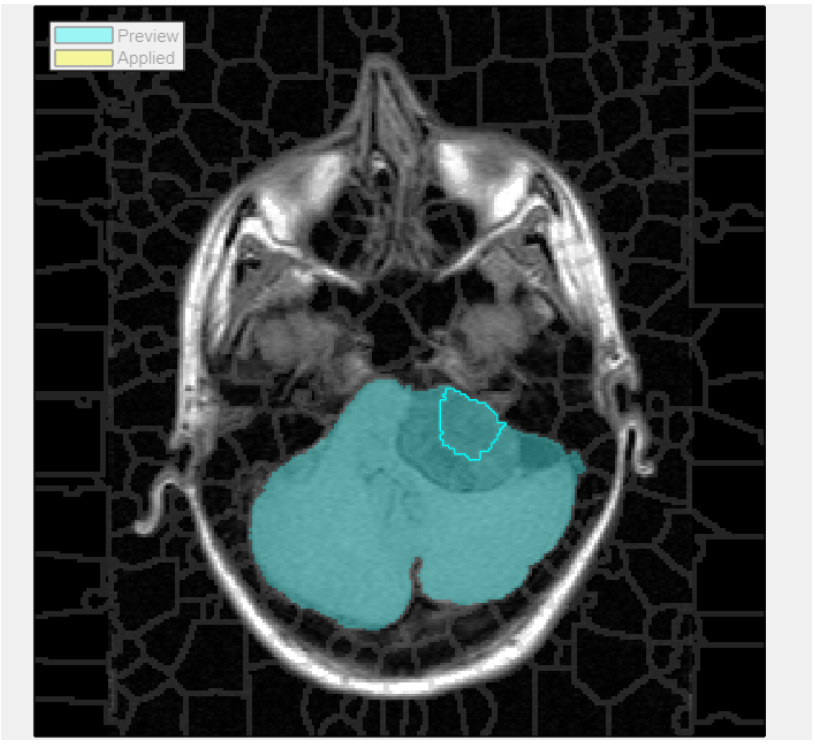
Otro método para umbralizar la imagen más cómodamente podría ser usar la aplicación image segmenter desde Matlab, que nos da la oportunidad de trabajar con la imagen de manera más cómoda de segmentar la imagen, en esta aplicación se puede importar la imagen a estudiar y desde ahí nos da acceso a múltiples herramientas para hacer una umbralización

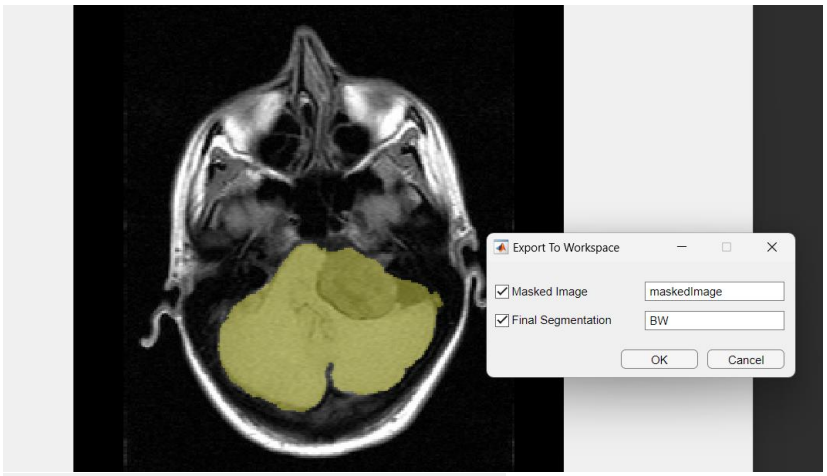




Se pueden usar múltiples funciones para umbralizar, de manera muy visual y facil, de los cuales se automatizan métodos que ya se han visto en esta serie de memorias como lo pueden ser el **superpixel, threshold, autocluster,** etc. Donde hasta se pueden obtener la imagen binaria o hasta la función para obtener los umbrales exactos usados.

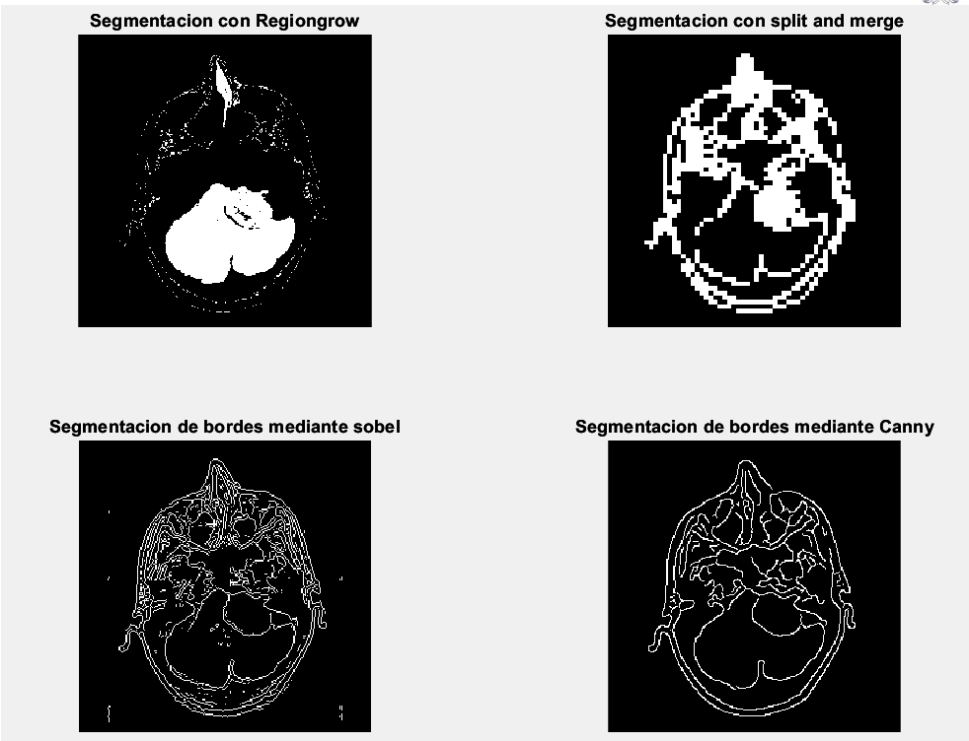
En esta imagen se muestran los superpixeles siendo marcados para rellenar todo el cerebro y la masa cerebral.





2. Con la imagen (o imágenes, en caso de emplear varios algoritmos) obtenida en el apartado anterior, realizar un filtrado morfológico que permita obtener una mejora (eliminar pequeños objetos indeseados, rellenar objetos, suavizar los bordes del objeto, …) del resultado obtenido por el algoritmo de segmentación utilizado. Puede utilizar la función de Matlab llamada bwmorph(…).

Antes de empezar vamos a recordar las segmentaciones obtenidas para tener una referencia de como son antes del postprocesado.



Explicación Uso de BW2 = bwmorph(imagen, OPERATION) :
El filtrado morfológico es una técnica de procesamiento de imágenes que utiliza operaciones morfológicas para transformar una imagen. Las operaciones morfológicas son operaciones que se basan en la forma y el tamaño de los objetos en una imagen.

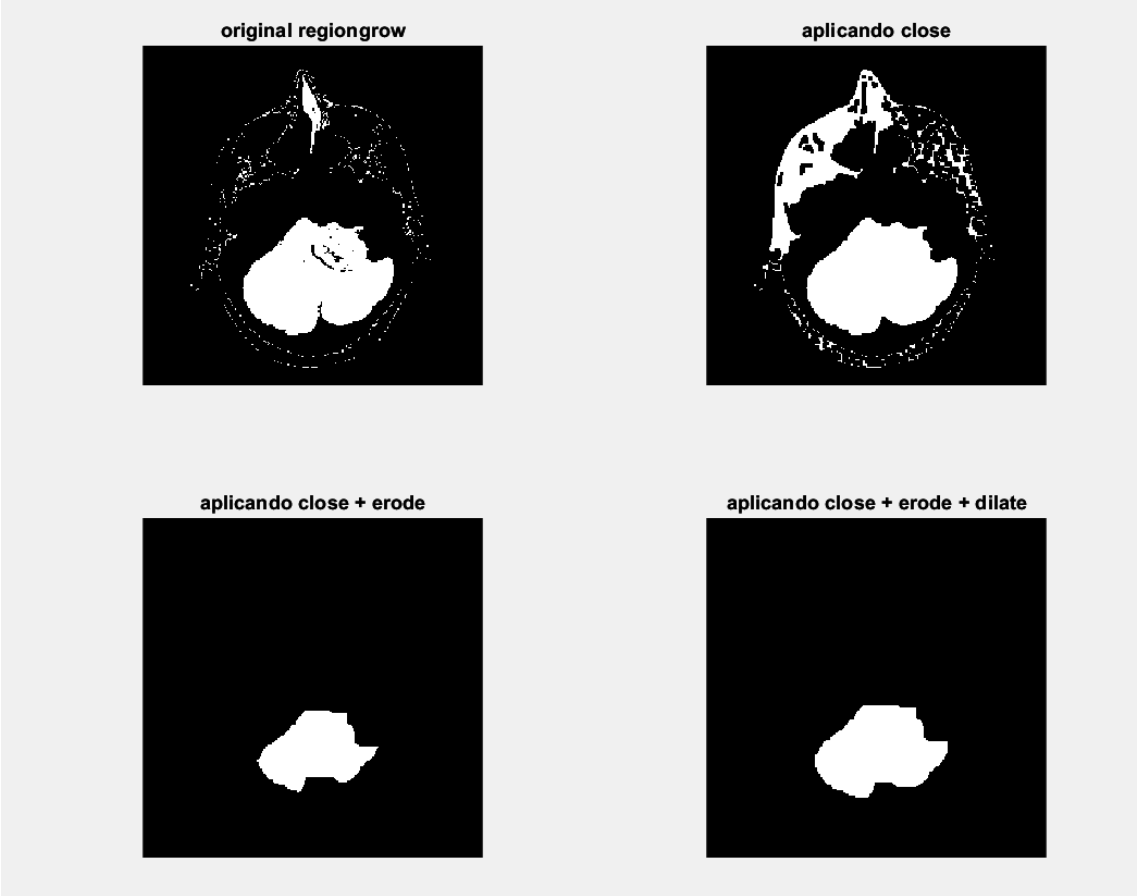
La operación de cierre es una operación morfológica que se utiliza para rellenar los huecos pequeños en los objetos de una imagen. La operación de cierre se realiza aplicando una operación de dilatación seguida de una operación de erosión.

- OPERATION es una cadena o vector de caracteres que puede tener uno de estos valores:
- **'bothat'** Resta la imagen de entrada de su cierre
- **'branchpoints'** Encuentra puntos de ramificación del esqueleto
- **'bridge'** Puente de píxeles previamente desconectados
- **'clean'** Elimina píxeles aislados (1 rodeados por 0)
- **'close'** Hace la operación de cierre consiste en una dilatación y una erosión (en ese orden)
- **'diag'** Relleno diagonal para eliminar la conectividad 8 de fondo
- **'endpoints'** Encuentra los puntos finales del esqueleto
- **'fill'** Rellena píxeles interiores aislados (0 rodeados por 1)
- **'hbreak'** Elimina los píxeles conectados en H
- **'majority'** Establece un píxel en 1 si hay cinco o más píxeles en su La vecindad de 3 por 3 son 1
- **'open'** Hace la operación de apertura consiste en una erosión y una dilatación (en ese orden)
- **'remove'** Establece un píxel en 0 si tiene 4 vecinos conectados que son todos unos, por lo que solo queda el límite píxeles
- Entre otros más (usar la función help bwmorph)

Codigo de Matlab:

```
%% 2. filtrado morfologico
%2.1 regiongrow
cerrado = bwmorph(regiongrow_ejemplo,"close",1); % Cierre
erosionado = bwmorph(cerrado,"erode",9); % Erosión
limpieza = bwmorph(erosionado,"dilate",4); % dilatado
%bordes =bwmorph(limpieza,"remove",1);

figure('Name', 'antes y despues')
subplot(2,2,1);
imshow(regiongrow_ejemplo)
title('original regiongrow')
subplot(2,2,2);
imshow(cerrado)
title('aplicando close')
subplot(2,2,3);
imshow(erosionado)
title('aplicando close + erode')
subplot(2,2,4);
imshow(limpieza)
title('aplicando close + erode + dilate')
```

Resultado:



Explicación:

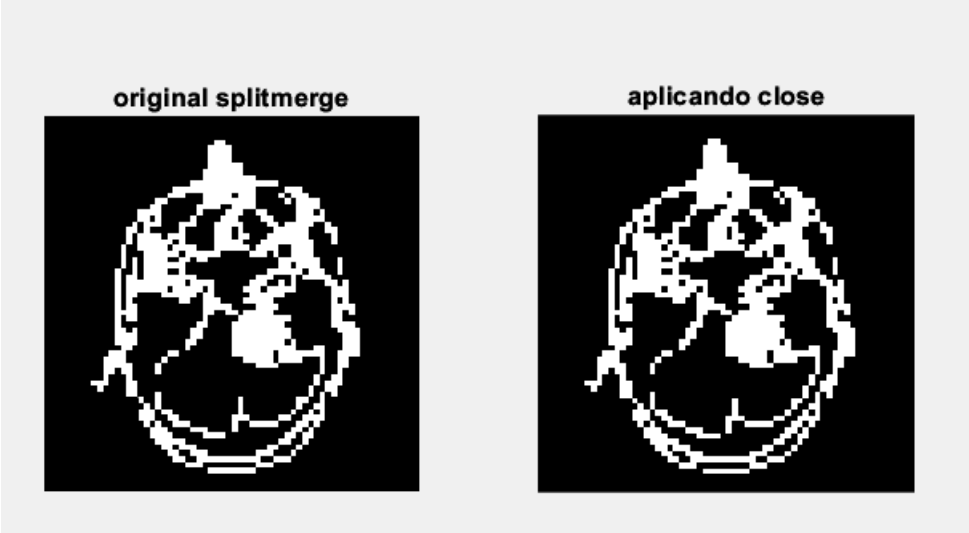
Se probaron distintos filtros morfológicos buscando eliminar los pixeles remanentes del algoritmo crecimiento en regiones, se empezó probando el close como fue recomendado durante la hora del laboratorio junto a otros métodos pero vi que volvían a generar parte del cráneo, no obstante el operando close es muy útil para rellenar el tumor

donde había pixeles faltantes, a partir de ahí se configuro manualmente un operando close, es decir una capa de erosión a la cual le dimos relativa fuerza para eliminar todos los pixeles que no se encontraran al objeto de interés, pero vimos una reducción de información de dicho objeto, por lo cual se tuvo que dilatar para obtener su tamaño y forma aproximada. No es la forma más pulcra de trabajar con los filtros morfológicos, pero fueron los mejores resultados obtenidos en vista del apartado 3.

Codigo de Matlab:

```
%% 2.2 filtrado a splitmerge
cerrado_split = bwmorph(splitmerge_final,"close",1); % erosion

figure('Name', 'antes y despues')
subplot(1,2,1);
imshow(splitmerge_final)
title('original splitmerge')
subplot(1,2,2);
imshow(cerrado_split)
title('aplicando close')
```

Resultado:



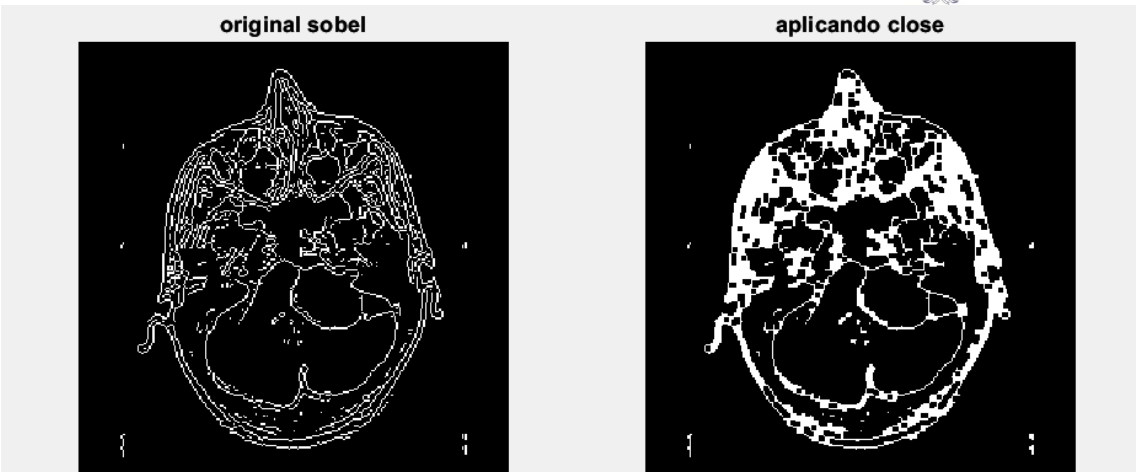
Explicación:

En este caso el filtro morfológico no puede hacer un gran cambio, esto es porque la umbralización conseguida no separa mucho el cráneo del cerebro y el tumor. Al aplicarle un filtro morfológico no sé puede hacer un gran cambio porque termina borrando toda la foto.

Codigo de Matlab:

```
%% 2.3 Filtrado sobel manual
cerrado_sobel = bwmorph(sobel_manual,"close",2); % erosion y dilatado

figure('Name', 'antes y despues')
subplot(1,2,1);
imshow(sobel_manual)
title('original sobel')
subplot(1,2,2);
imshow(cerrado_sobel)
title('aplicando close')
```

Resultado:

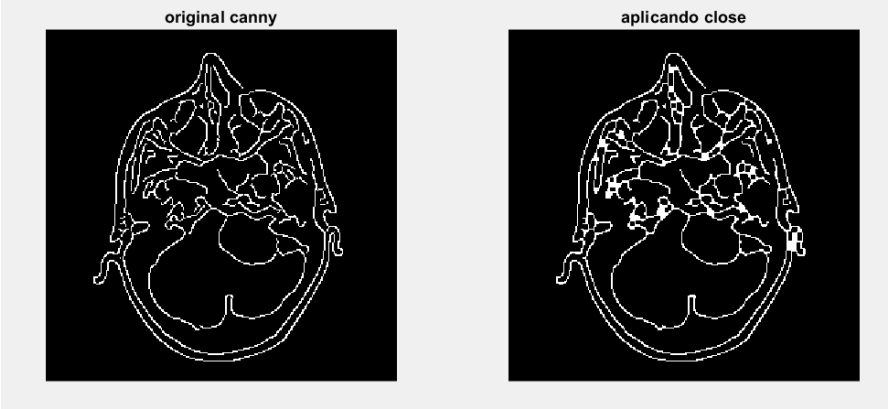


Explicación:
Este caso es parecido al anterior donde los filtros morfológicos no pueden modificar parte de la imagen sin afectar el objeto de estudio, aquí se dejo el filtro de cierre como ejemplo.

Codigo de Matlab:

```
%% 2.4 Filtrado canny
cerrado_sobel = bwmorph(canny_seg,"close",2); % erosion y dilatado


figure('Name', 'antes y despues')
subplot(1,2,1);
imshow(canny_seg)
title('original canny')
subplot(1,2,2);
imshow(cerrado_sobel)
title('aplicando close')
```

Resultado:

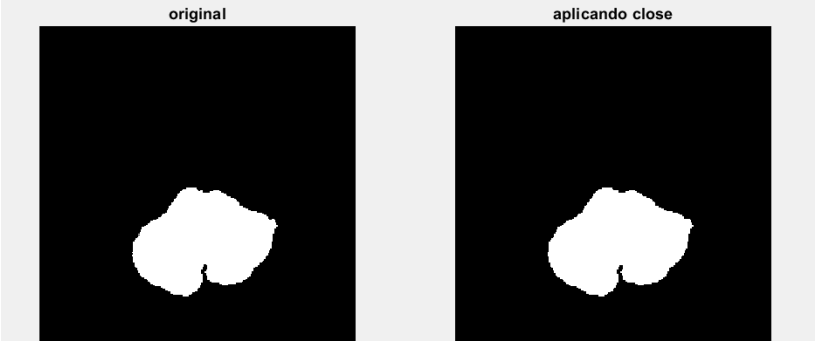


Explicación:
En este caso también pasa con el canny, que la aplicación de un filtro morfológico ayuda a darle retoques al tumor y a la masa cerebral, sin embargo, no es una forma de umbralizarlo porque al aplicar dichos filtros como el de erosión para eliminar el cráneo, nos cargamos también el cerebro y el tumor.

Codigo de Matlab:

```
%% Imagenes trabajadas en APP Image Segmenter
cerrado_BW = bwmorph(BW,"close",2); % erosion y dilatado

figure('Name', 'antes y despues')
subplot(1,2,1);
imshow(BW)
title('original')
subplot(1,2,2);
imshow(cerrado_BW)
title('aplicando close')
```

Resultado:



Explicación:

En este caso el filtro morfológico ayudo bastante a tener mejor la imagen, teniendo algunos contornos más redondeados lo que facilita su comprensión, además de redondear los picos que probablemente hubieran sido errores de la umbralización dejando una imagen más suave sin ruido.

Se podría concluir que el mejor algoritmo fue el regiongrow y el uso de superpixeles donde se pudo separar el cráneo sin perder la forma del cerebro ni del tumor.

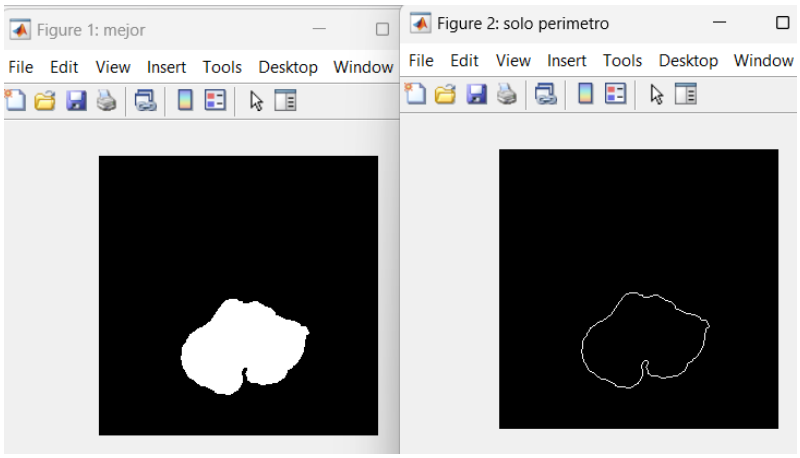
3. Obtener descriptores básicos de cada uno de los objetos de interés, entre ellos el área, perímetro y compacidad. Como parámetro opcional, se propone la obtención de la firma del contorno [1-3] y de algunos momentos básicos (media, desviación típica, skewness, …). Puede utilizar la función de Matlab llamada regionprops(…).

Codigo de Matlab:

```
%% Descriptores
%area
figure('Name', 'mejor')
imshow(cerrado_BW)
a=regionprops(cerrado_BW,'Area');
areaa=a.Area %6926

%perimetro
p=regionprops(cerrado_BW,'Perimeter');
perimetro=p.Perimeter %351.4090

regiongrow_perimetro = bwmorph(cerrado_BW,"remove",1); % dilatado
figure('Name', 'solo perimetro')
imshow(regiongrow_perimetro)
```

Resultado:

```
areaa =

        6926


perimetro =

  351.4090
```



Explicación:
Aqui se puede ver cómo sería el perímetro del objeto de estudio usando el filtro morfológica remove.

Codigo de Matlab:

```
%compacidad
Compacidad=(perimetro.^2)/areaa
```

Resultado:

```
Compacidad =

   17.8297
```

Codigo de Matlab:
Media=mean2(`cerrado_BW`*);*

Resultado:

```
Media =

    0.1057
```

Explicación:
Consideramos que es una media aceptable en función de nuestra región de interés.

Codigo de Matlab:

```
%desviacion tipica
Desviacion=std2(cerrado_BW)
```

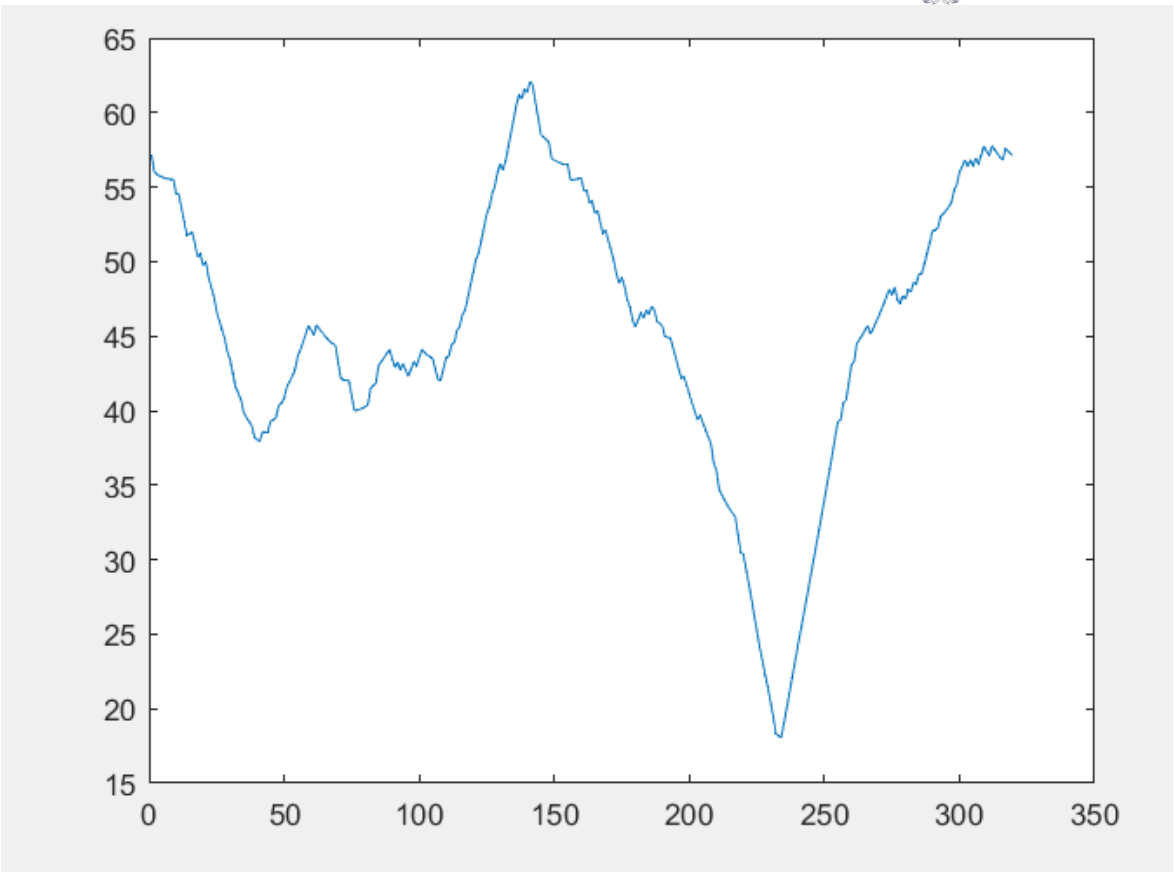
Resultado:

```
Desviacion =

    0.3074

ans =

  struct with fields:

          Area: 6926
      Centroid: [132.5176 176.0027]
   BoundingBox: [75.5000 131.5000 118 88]
```

Codigo de Matlab de firma de contorno:

```
%firma de contorno
center=regionprops(cerrado_BW,'Centroid');
boundary=bwboundaries(cerrado_BW);
for i=1: length(center)
    c=center(i).Centroid;
    b=boundary(i);
    x=b{1,1}(:,1);
    y=b{1,1}(:,2);
    d=sqrt((y-c(1)).^2+(x-c(2)).^2);
    t=1:1:length(d);
    figure(i);
    plot(t,d)
end
```

Resultado:



Explicación:

Los **descriptores básicos** son características simples que se pueden extraer de una imagen para representar su contenido visual. Se utilizan en una variedad de tareas de procesamiento de imágenes, incluyendo:

- **Reconocimiento de objetos**: Los descriptores básicos se pueden utilizar para identificar objetos en una imagen. Por ejemplo, se pueden utilizar para identificar personas, vehículos o productos.
- **Segmentación de objetos**: Los descriptores básicos se pueden utilizar para segmentar objetos en una imagen. Esto significa dividir la imagen en regiones diferentes, cada una de las cuales contiene un objeto.
- **Medición de características**: Los descriptores básicos se pueden utilizar para medir características de los objetos en una imagen. Por ejemplo, se pueden utilizar para medir el tamaño, la forma o la orientación de un objeto.

Algunos ejemplos de los descriptores obtenidos son:

- **Área:** El área de una imagen es la cantidad de píxeles que contiene. El área de una imagen es una medida de su tamaño.
- **Perímetr**o: El perímetro de una imagen es la longitud de su borde. Se calcula sumando la longitud de cada píxel en el borde de la imagen.

- **Compacidad**: La compacidad de una imagen es una medida de su forma. Una imagen compacta tiene un perímetro corto en comparación con su área.
- **Firma del contorno**: es una representación matemática de su forma. Se calcula a partir de los valores de intensidad de los píxeles a lo largo del contorno de la imagen. Se puede calcular utilizando una variedad de métodos, arriba está el algoritmo para *calcular la firma de contorno de distancia al centroide*

Momentos básicos

Los momentos básicos son una medida de la distribución de la intensidad de los píxeles en una imagen. Los momentos básicos incluyen la media, la desviación estándar, la asimetría y la curtosis.

- **Media**: La media de una imagen es el valor promedio de la intensidad de los píxeles en la imagen.
- **Desviación estándar**: La desviación estándar de una imagen es una medida de la dispersión de los valores de intensidad de los píxeles alrededor de la media.

La curtosis y asimetría no las calcule, pero también se podrían hacer.

Estos son los descriptores básicos para el objeto de interés tomando el tumor y masa cerebral, se podría configurar para segmentar solamente la masa cerebral o solamente el tumor, pero me parece que sería repetir nuevamente muchos métodos cuando ya se vio en el crecimiento de regiones como era repetir el procedimiento para hacer lo mismo. Preferí obtener todas las funciones opcionales y curiosas para obtener los descriptores.

Conclusión

La segmentación de imágenes es una herramienta vital que debe estar en el conocimiento de cada ingeniero de la salud, en este trabajo se pudo ver como aplicando los conocimientos obtenidos en cada practica realizada se pudo superar los retos conseguidos, además del estudio de fuentes como puede ser la búsqueda de información en la api de Matlab, aprender a modificar códigos ya existentes en libros como puede ser el recomendado "Digital Image Processing" de Gonzalez R.C además de ver el canal de YouTube de Matlab, pudiendo lograr ver distintos métodos de segmentación como traen distintos resultados para una misma imagen.

La segmentación es un proceso fundamental en el análisis de imágenes por computadoras o Inteligencia artificial. Permite dividir una imagen en regiones o segmentos, cada uno de los cuales representa un objeto o una característica de la imagen. Esta división puede facilitar el análisis de la imagen, ya que permite tratar cada segmento como una entidad independiente.

En general, los filtros morfológicos son una herramienta poderosa que se puede utilizar para una variedad de tareas de procesamiento de imágenes. En este trabajo se vio inicialmente para postprocesados, pero Tambien pueden ser aplicados directamente para la segmentación.

Muchas gracias por seguir el trabajo hasta aquí, Espero que pueda haber aprendido tanto como yo.

Bibliografía recomendada

- Gonzalez, R.C., Woods, R.E. (2007). Digital Image Processing, 3ed. Prentice-Hall.
- Gonzalez, R.C., Woods, R.E., Eddins, S.L. (2009). Digital Image Processing using MATLAB, 2ed. Prentice-Hall.
- Pratt, W.K. (2007). Digital Image Processing, 4ed. Wiley.
- Chan, T.F., Vese, L. (2001). Active Contours Without Edges, IEEE Transactions on Image Processing, vol 10, no 2, pp. 266-276.
- Guion de laboratorio para la practica 4 de ingeniería de la salud

Y gracias a las clases de Enrique Nava Baro profesor de imágenes biomédicas en la universidad de Málaga

Bibliografía especificada

1. Copyright 2002-2004 R. C. Gonzalez, R. E. Woods, & S. L. Eddins Digital Image Processing Using MATLAB, Prentice-Hall, 2004 Revision: 1.4 Date: 2003/10/26 22:35:37

Código: (ver mejor el propio archivo de Matlab en la carpeta zip)

```
%Cargamos la imagen y la ajustamos automaticamente
%Borramos las variables cargadas y cargamos la imagen
clear variables
clc
close all

%leemos la imagen para tener referencia
imagen=dicomread('im5'); %imagen asignada (sin procesar)
imagen_ajustada=imadjust(imagen); %ajuste automatico
%comparacion
figure('Name', 'Observación de la imagen previa a la practica')
subplot(1,2,1);
imshow(imagen)
title('imagen 5 sin procesar')
subplot(1,2,2);
imshow(imagen_ajustada)
title('imagen 5 ajustada')
%% Utilizar uno o varios de los algoritmos de segmentación de imágenes
%1.1 regiongrow usando S como escalar (regiongrow(f, S, T))
%imagen con 250 semillas y 75 umbral
[masa_cerebral, NR1, SI1, TI1] = regiongrow(imagen, 250, 75);
%imagen con 270 semillas y 120 umbral
[solo_craneo, NR2, SI2, TI2] = regiongrow(imagen, 270, 120);
%imagen con 250 semillas y 120 umbral
[craneo_y_cerebro, NR3, SI3, TI3] = regiongrow(imagen, 250, 120);

zona_interes = craneo_y_cerebro - solo_craneo;

figure('Name', 'Comparación de distintas semillas para regiongrow')
subplot(2,2,1);
imshow(masa_cerebral)
title('imagen con masa cerebral')
subplot(2,2,2);
imshow(solo_craneo)
title('imagen con solo craneo')
subplot(2,2,3);
imshow(craneo_y_cerebro)
title('imagen de craneo y zona de interes')
subplot(2,2,4);
imshow(zona_interes)
title('zona de interes')
regiongrow_ejemplo=zona_interes;
%% 1.2 splitmerge v = split_test(B, mindim, fun)
%predicate(imagen)
splitmerge_final = splitmerge(imagen, 4, @predicate);
imshow(splitmerge_final)

%se recomienda probar valores distintos de mindin y tambien modificar la
%funcion predicate especificamente los valores k y M

%% 1.3 Utilización de algoritmos de detección y extracción de bordes y
contornos de la imagen
%1 Filtro sobel
[imagen_binaria,thresh]= edge(imagen, 'sobel'); %usa sobel por defecto
thresh; %ver umbral automatico
```

```
sobel_manual= edge(imagen, 'sobel',0.00035);

figure('Name', 'Visualización para función edge')
subplot(1,2,1);
imshow(imagen_binaria)
title('Edge con sobel usando umbral automatico')
subplot(1,2,2);
imshow(sobel_manual)
title('Edge con sobel usando umbral manual')
%codigo del libro:
%g = sqrt(imfilter(imagen,fs,'replicate').^2 +
imfilter(imagen,fs','replicate').^2);

%% 1.3.2 (filtro canny)

[imagen_automatica,thresh]= edge(imagen, 'canny');
thresh; %ver umbral automatico
%imshow(imagen_automatica)

%canny manual
canny_seg= edge(imagen, 'canny', [0 0.1800],sqrt(3));


figure('Name', 'Visualización canny')
subplot(1,2,1);
imshow(imagen_automatica)
title('Imagen ajustada automaticamente')
subplot(1,2,2);
imshow(canny_seg)
title('canny con unos umbrales manuales')

%% Recapitulación de imagenes finales

figure('Name', 'Comparación de distintos algoritmos de segmentacion usados')
subplot(2,2,1);
imshow(regiongrow_ejemplo)
title('Segmentacion con Regiongrow')

subplot(2,2,2);
imshow(splitmerge_final)
title('Segmentacion con split and merge')

subplot(2,2,3);
imshow(sobel_manual)
title('Segmentacion de bordes mediante sobel')

subplot(2,2,4);
imshow(canny_seg)
title('Segmentacion de bordes mediante Canny')

%% Imagenes trabajadas en APP Image Segmenter
figure('Name', 'Cerebro')
imshow(BW)
title('Segmentacion conseguida')
%% 2. filtrado morfologico
%2.1 regiongrow
cerrado = bwmorph(regiongrow_ejemplo,"close",1); % Cierre
erosionado = bwmorph(cerrado,"erode",9); % Erosión
limpieza = bwmorph(erosionado,"dilate",4); % dilatado
```

```
%bordes =bwmorph(limpieza,"remove",1);

figure('Name', 'antes y despues')
subplot(2,2,1);
imshow(regiongrow_ejemplo)
title('original regiongrow')
subplot(2,2,2);
imshow(cerrado)
title('aplicando close')
subplot(2,2,3);
imshow(erosionado)
title('aplicando close + erode')
subplot(2,2,4);
imshow(limpieza)
title('aplicando close + erode + dilate')

regiongrow_final=limpieza;
%% 2.2 filtrado a splitmerge
cerrado_split = bwmorph(splitmerge_final,"close",1); % erosion

figure('Name', 'antes y despues')
subplot(1,2,1);
imshow(splitmerge_final)
title('original splitmerge')
subplot(1,2,2);
imshow(cerrado_split)
title('aplicando close')

%% 2.3 Filtrado sobel manual
cerrado_sobel = bwmorph(sobel_manual,"close",2); % erosion y dilatado

figure('Name', 'antes y despues')
subplot(1,2,1);
imshow(sobel_manual)
title('original sobel')
subplot(1,2,2);
imshow(cerrado_sobel)
title('aplicando close')

%% 2.4 Filtrado canny
cerrado_sobel = bwmorph(canny_seg,"close",2); % erosion y dilatado


figure('Name', 'antes y despues')
subplot(1,2,1);
imshow(canny_seg)
title('original canny')
subplot(1,2,2);
imshow(cerrado_sobel)
title('aplicando close')

%% Imagenes trabajadas en APP Image Segmenter
cerrado_BW = bwmorph(BW,"close",2); % erosion y dilatado

figure('Name', 'antes y despues')
subplot(1,2,1);
imshow(BW)
title('original')
subplot(1,2,2);
```

```
imshow(cerrado_BW)
title('aplicando close')

%% Descriptores
%area
figure('Name', 'mejor')
imshow(cerrado_BW)
a=regionprops(cerrado_BW,'Area');
areaa=a.Area %6926

%perimetro
p=regionprops(cerrado_BW,'Perimeter');
perimetro=p.Perimeter %351.4090

regiongrow_perimetro = bwmorph(cerrado_BW,"remove",1); % dilatado
figure('Name', 'solo perimetro')
imshow(regiongrow_perimetro)

%compacidad
Compacidad=(perimetro.^2)/areaa

%% Otros descriptores
Media=mean2(cerrado_BW)

%desviacion tipica
Desviacion=std2(cerrado_BW)

%skewness
S=skewness(cerrado_BW);

%firma de contorno
center=regionprops(cerrado_BW,'Centroid');
boundary=bwboundaries(cerrado_BW);
for i=1: length(center)
       c=center(i).Centroid;
       b=boundary(i);
       x=b{1,1}(:,1);
       y=b{1,1}(:,2);
       d=sqrt((y-c(1)).^2+(x-c(2)).^2);
       t=1:1:length(d);
       figure(i);
       plot(t,d)
end

regionprops(cerrado_BW)
%% Funciones

%1funcion dada por el profe basada en un umbral sobre el nivel de gris
(González)
function [g, NR, SI, TI] = regiongrow(f, S, T)
f = double(f);
if numel(S) == 1
   SI = f == S;
   S1 = S;
else
   SI = bwmorph(S, 'shrink', Inf);
   J = SI;
   S1 = f(J); % Array of seed values.
end
```

```
TI = false(size(f));
for K = 1:length(S1)
   seedvalue = S1(K);
   S = abs(f - seedvalue) <= T;
   TI = TI | S;
end

[g, NR] = bwlabel(imreconstruct(SI, TI));
end

%1.1.2
function g = splitmerge(f, mindim, fun)
Q = 2^nextpow2(max(size(f)));
[M, N] = size(f);
f = padarray(f, [Q - M, Q - N], 'post');

S = qtdecomp(f, @split_test, mindim, fun);
Lmax = full(max(S(:)));
g = zeros(size(f));
MARKER = zeros(size(f));
% Begin the merging stage.
for K = 1:Lmax
   [vals, r, c] = qtgetblk(f, S, K);
   if ~isempty(vals)
      for I = 1:length(r)
         xlow = r(I); ylow = c(I);
         xhigh = xlow + K - 1; yhigh = ylow + K - 1;
         region = f(xlow:xhigh, ylow:yhigh);
         flag = feval(fun, region);
         if flag
            g(xlow:xhigh, ylow:yhigh) = 1;
            MARKER(xlow, ylow) = 1;
         end
      end
   end
end

% Finally, obtain each connected region and label it with a
% different integer value using function bwlabel.
g = bwlabel(imreconstruct(MARKER, g));

% Crop and exit
g = g(1:M, 1:N);
end

function v = split_test(B, mindim, fun)
k = size(B, 3);
v(1:k) = false;
for I = 1:k
   quadregion = B(:, :, I);
   if size(quadregion, 1) <= mindim
      v(I) = false;
      continue
   end
   flag = feval(fun, quadregion);
   if flag
      v(I) = true;
   end
```

```
end
end

%1.1.3 mejor resultado
function flag = predicate(region)
sd = std2(region);
m = mean2(region);
k=2.4;
M=150;
flag = abs(M-m)<k*sd;
end

function [BW,maskedImage] = segmentImage(X)
%segmentImage Segment image using auto-generated code from Image Segmenter
app
%  [BW,MASKEDIMAGE] = segmentImage(X) segments image X using auto-generated
%  code from the Image Segmenter app. The final segmentation is returned in
%  BW, and a masked image is returned in MASKEDIMAGE.

% Auto-generated by imageSegmenter app on 24-Nov-2023
%----------------------------------------------------


% Adjust data to span data range.
X = imadjust(X);

% Create empty mask
BW = false(size(X,1),size(X,2));

% Draw ROIs

xPos = [119.3727 117.6341 116.3302 110.2453 106.3336 105.4643 104.1604
103.2912 102.4219 101.9873 101.5526 101.1180 99.8141 99.3795 98.9448 98.0756
97.2063 95.9024 95.4677 95.4677 95.0331 94.5985 93.2946 93.2946 92.8599
92.4253 91.5560 91.5560 91.1214 91.1214 90.2521 90.2521 89.3829 88.9482
88.5136 87.2097 86.7750 86.3404 85.9058 84.1672 82.4287 81.9941 80.6902
80.2555 80.2555 80.2555 80.2555 80.2555 80.2555 79.8209 79.8209 79.8209
79.3862 78.5170 78.0823 78.0823 78.0823 78.0823 78.0823 78.0823 78.0823
78.0823 78.0823 78.0823 78.0823 78.0823 78.0823 78.0823 78.0823 78.0823
78.0823 78.0823 78.0823 78.0823 78.0823 78.0823 78.0823 78.0823 78.0823
78.0823 78.0823 78.0823 78.0823 78.5170 79.3862 80.2555 80.2555 82.4287
84.1672 84.6019 85.0365 86.7750 88.0789 88.9482 88.9482 90.2521 91.1214
92.4253 93.7292 95.0331 95.4677 96.7716 97.6409 98.0500 99.3795 101.1180
101.9873 104.1604 104.5951 105.4643 106.3336 106.3336 106.7683 107.6375
108.0722 108.5068 108.9414 110.2453 110.6800 111.9839 112.4185 112.8531
115.0263 116.7649 118.5034 118.9380 119.3727 121.1112 121.5458 121.9805
123.2844 123.7190 125.0229 125.4576 125.8922 126.3268 128.0654 130.2385
130.2385 130.2385 130.6732 131.5424 131.9771 132.4117 132.4117 132.8463
134.1503 134.5849 134.5849 135.0195 135.8888 136.7581 137.1927 138.0620
138.4966 138.9312 139.3659 140.6698 141.1044 142.4083 143.2776 144.5815
145.0161 145.4508 145.8854 147.1893 147.6239 148.0586 148.9278 149.7971
151.1010 151.9703 153.7088 154.5781 155.8820 156.3166 157.6205 158.4898
160.6630 161.0976 161.9669 162.8362 163.2708 164.5747 165.0093 165.4440
166.7479 168.4864 168.9211 169.3557 169.7903 171.0942 171.5289 173.2674
173.7020 174.1367 175.4406 175.4406 175.8752 176.3098 177.6138 178.0484
178.4830 179.7869 180.2216 182.3947 182.8294 183.6986 184.1333 184.5679
185.8718 186.7411 187.1757 188.0450 188.4796 188.9143 189.3489 190.6528
191.0874 191.5221 192.3913 192.8260 192.8260 193.2606 193.2606 193.2606
193.6952 193.6952 193.6952 193.6952 194.5645 194.5645 194.5645 194.5645
```

```
194.9992 194.9992 194.9992 194.9992 194.9992 195.4338 195.4338 195.4338
195.4338 195.4338 195.4338 195.4338 195.4338 194.5645 193.2606 192.8260
192.3913 191.5221 191.0874 191.0874 190.6528 190.2182 188.0450 187.1757
186.7411 186.7411 185.8718 185.0025 184.1333 183.6986 182.3947 181.5255
179.7869 178.4830 178.0484 178.0484 177.1791 175.8752 175.0059 173.2674
171.5289 171.0942 169.7903 169.7903 169.3557 169.3557 169.7903 171.0942
171.5289 171.5289 171.9635 171.9635 171.5289 171.0942 170.6596 169.7903
169.3557 167.6171 167.1825 165.4440 165.0093 164.5747 162.8362 162.4015
161.9669 160.6630 160.2284 159.7937 158.9244 158.4898 157.6205 156.7513
156.3166 155.8820 155.4474 154.5781 154.1435 153.7088 152.4049 151.9703
150.2317 149.7971 149.3625 148.9278 147.6239 147.1893 145.8854 145.0161
143.7122 143.2776 141.1044 140.2351 138.9312 138.4966 137.1927 136.7581
135.8888 134.5849 132.8463 132.4117 131.9771 130.6732 130.2385 129.8039
128.0654 127.6307 125.8922 125.4576 124.1537 123.7190 122.8497 121.5458
121.1112 119.3727 118.9380 117.6341 117.1995 116.7649 116.3302 115.4610
115.0263 115.8693];
yPos = [135.0195 135.0195 135.4542 138.0620 139.8005 140.2351 141.1044
141.5390 141.5390 141.5390 141.9737 143.2776 143.7122 144.1469 145.4508
145.8854 146.3200 147.6239 148.0586 148.4932 149.7971 150.2317 151.1010
152.4049 152.8396 154.5781 155.0127 156.3166 156.7513 157.1859 157.6205
158.4898 158.4898 158.9244 159.3591 159.7937 159.7937 160.6630 161.0976
161.5323 161.9669 162.8362 163.2708 163.2708 163.7054 164.1401 165.0093
165.4440 165.8786 167.1825 167.6171 168.4864 169.7903 172.8328 174.1367
176.7445 178.0484 178.4830 178.9177 179.3523 180.2216 180.6562 181.5255
182.3947 182.8294 183.2640 183.6986 184.5679 185.0025 185.8718 186.7411
187.1757 187.6104 189.7835 190.2182 191.5221 191.9567 192.3913 193.2606
193.6952 194.5645 195.8684 196.3031 197.6070 198.0416 198.0416 198.4762
200.2148 200.6494 201.9533 202.8226 204.1265 204.5611 204.9958 205.4304
206.2997 206.7343 206.7343 207.1689 208.4728 208.9075 208.9075 209.3421
209.4444 209.7767 211.0806 211.5153 212.8192 213.2538 213.2538 213.2538
213.6885 214.1231 215.4270 215.4270 215.4270 215.8616 216.2963 217.1655
217.6002 217.6002 217.6002 217.6002 218.0348 218.0348 218.4694 218.4694
219.7733 219.7733 219.7733 219.7733 219.7733 219.7733 219.7733 219.7733
219.7733 219.7733 219.3387 218.4694 218.0348 218.0348 217.6002 216.2963
215.4270 214.9924 214.9924 213.6885 213.6885 213.2538 213.2538 213.2538
213.2538 212.8192 212.8192 211.9499 211.9499 211.9499 211.9499 211.9499
211.9499 211.9499 211.9499 211.9499 211.9499 211.9499 212.8192 212.8192
213.2538 213.2538 213.2538 213.2538 213.2538 213.2538 213.6885 214.9924
214.9924 214.9924 214.9924 214.9924 214.9924 214.9924 214.9924 214.9924
213.6885 213.2538 213.2538 212.8192 211.0806 211.0806 210.6460 209.7767
209.3421 208.9075 207.1689 206.7343 206.2997 206.2997 205.4304 205.4304
204.9958 204.5611 204.1265 203.2572 202.8226 202.3879 201.0840 200.2148
200.2148 199.7801 198.4762 198.0416 196.7377 196.3031 195.8684 195.8684
195.4338 194.1299 193.2606 191.9567 189.7835 189.3489 188.0450 187.1757
185.8718 183.6986 182.8294 182.8294 181.5255 181.0908 180.6562 178.9177
178.4830 177.1791 176.7445 176.3098 175.8752 174.5713 173.7020 171.9635
170.2250 169.7903 169.3557 167.6171 166.3132 165.8786 165.4440 165.0093
163.7054 163.2708 162.8362 161.9669 161.9669 161.9669 161.5323 161.5323
161.0976 161.0976 160.6630 160.6630 159.3591 158.9244 158.9244 157.6205
157.1859 156.7513 156.7513 155.4474 155.0127 155.0127 154.5781 154.5781
152.4049 152.4049 150.2317 148.9278 148.0586 147.6239 146.7547 146.7547
146.3200 145.4508 144.1469 143.7122 142.4083 141.5390 141.1044 141.1044
141.1044 141.1044 141.1044 141.1044 140.2351 139.8005 139.3659 139.3659
139.3659 139.3659 139.3659 139.3659 139.3659 139.3659 139.3659 139.3659
139.3659 139.3659 139.3659 139.3659 139.3659 139.3659 139.3659 139.3659
139.3659 139.3659 139.3659 139.3659 139.3659 139.3659 139.3659 139.3659
138.9312 138.0620 137.1927 137.1927 137.1927 135.8888 135.4542 135.4542
135.4542 135.4542 135.4542 135.4542 135.4542 135.4542 135.4542 135.4542
135.4542 135.4542 135.4542 135.0195 135.0195 135.0195 135.0195 135.0195
```

```
135.0195 134.5849 134.5849 134.5849 134.5849 134.5849 134.5849 134.5849
134.5849 134.5849 134.5849 134.5849 135.4771];
m = size(BW, 1);
n = size(BW, 2);
addedRegion = poly2mask(xPos, yPos, m, n);
BW = BW | addedRegion;

% Create masked image.
maskedImage = X;
maskedImage(~BW) = 0;
end
```