Visió per Computador

Homework 4: Pedretes

Huiwen Cao (**12F**)
Jordi Bru Carci (**12F**)
16/04/2023
2022-2023.Q2

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Tabla de contenidos

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

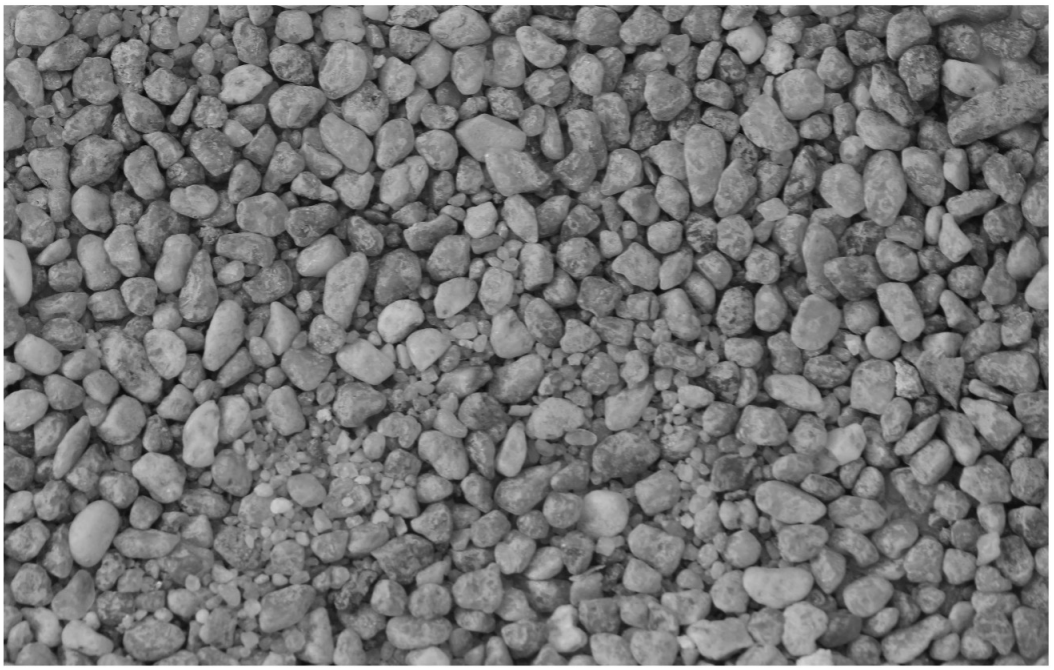
Introducción

En esta entrega se nos pide que creemos un programa capaz de contar el número de piedras en una imagen que cumpla las dimensiones que hemos decidido. Asimismo, el programa debe ser capaz de mostrar la misma imagen, pero con una superposición de las piedras segmentadas para resaltar las piedras resultantes.

Procedimiento

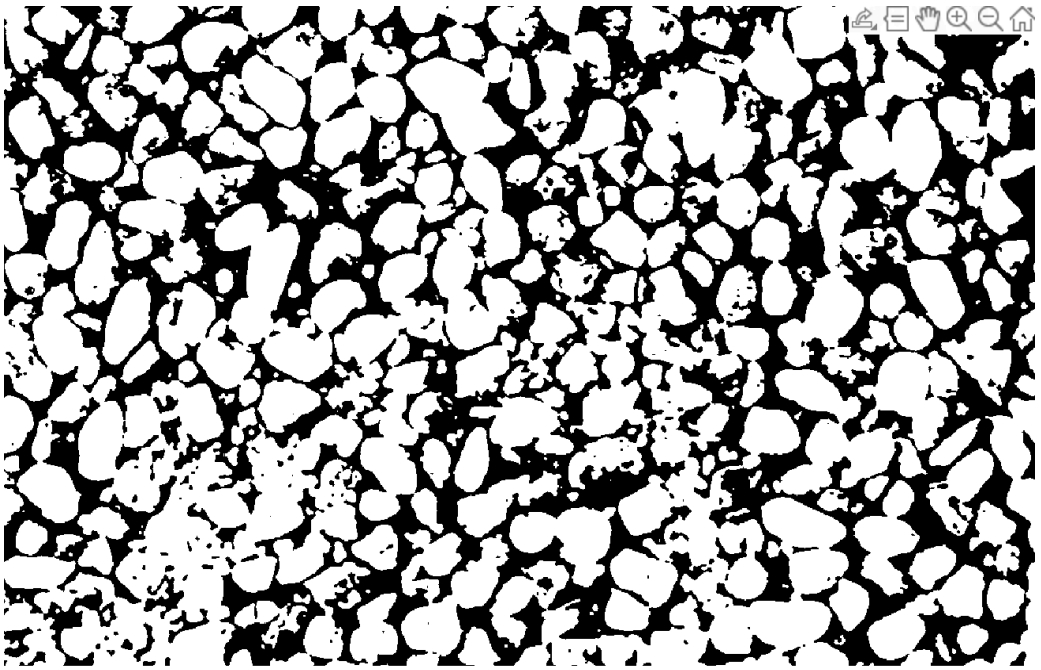
Primero de todo, obtenemos la imagen proporcionada por el enunciado para leerla y pasarla a escala de grises. A partir de ahí, le aplicaremos un filtro de mediana con una ventana de 4x4 para poder reducir el ruido que pueda haber.

```
I = rgb2gray(imread('imagen.jpg'));
imshow(I)

I = medfilt2(I,[4,4]); %reduir soroll
```



Una vez filtrado, binarizaremos la imagen filtrada para poder trabajar mejor con todos los objetos pequeños. Para ello, hemos optado por usar el método de umbralización de Otsu mediante la función graythresh() para definir el umbral que separará los píxeles de primer plano (piedras) de los de fondo.

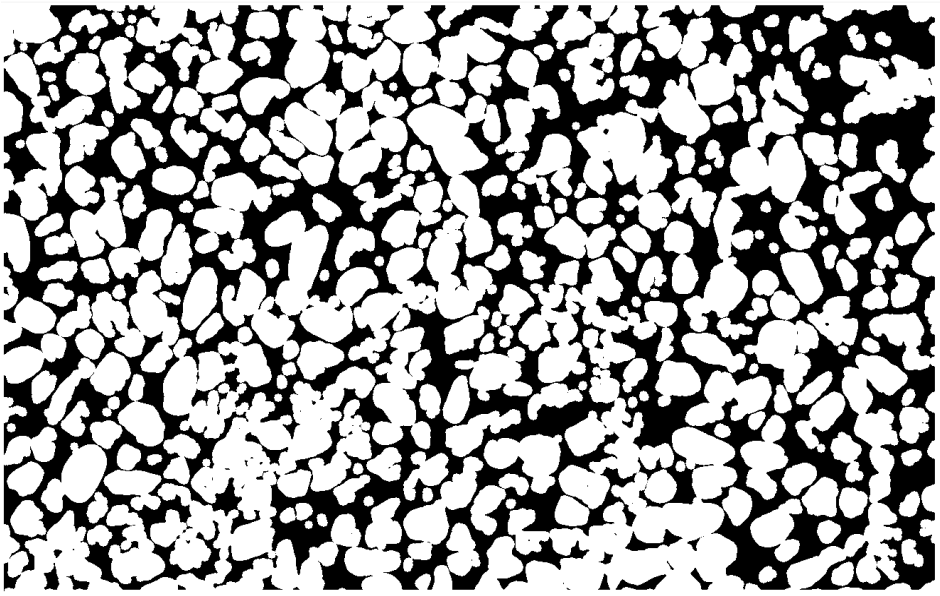
```
% Binarització de la imatge
level = 255*graythresh(I);
BW = I > level;
imshow(BW);
```



Antes de calcular y contar el número de piedras, primero debemos eliminar posibles elementos no deseados gracias al open (imopen()). Para ello decidimos crear un elemento estructurante con forma de esfera de un radio de 5px que creemos apropiado para el caso.

A continuación, aplicaremos una dilatación para poder conectar los píxeles que representan ser piedras para una mayor claridad. En este caso, usamos un elemento estructurante de ones(2,2) ya que nos ha dado mejores resultados.

```
BW = imopen(BW, strel("sphere", 5));
BW = imdilate(BW, ones(2,2));
imshow(BW)
```



Como se puede ver en la imagen binarizada, el resultado obtenido no es lo ideal debido a la presencia de objetos superpuestos y a la variabilidad en la apariencia, como cambios en la textura o forma. Esto hace difícil encontrar un umbral de binarización que sea adecuado para todos los casos.

En este punto del programa ya tenemos una imagen binarizada con la que creemos oportuno empezar a contar las piedras. Y para ello, primero decidimos el tamaño que queremos de dimensiones. Así pues, podremos llamar a nuestro método RangPedretes.m que nos facilitará una imagen binarizada a partir de la obtenida sin los elementos que no cumplan las condiciones predefinidas por nosotros.

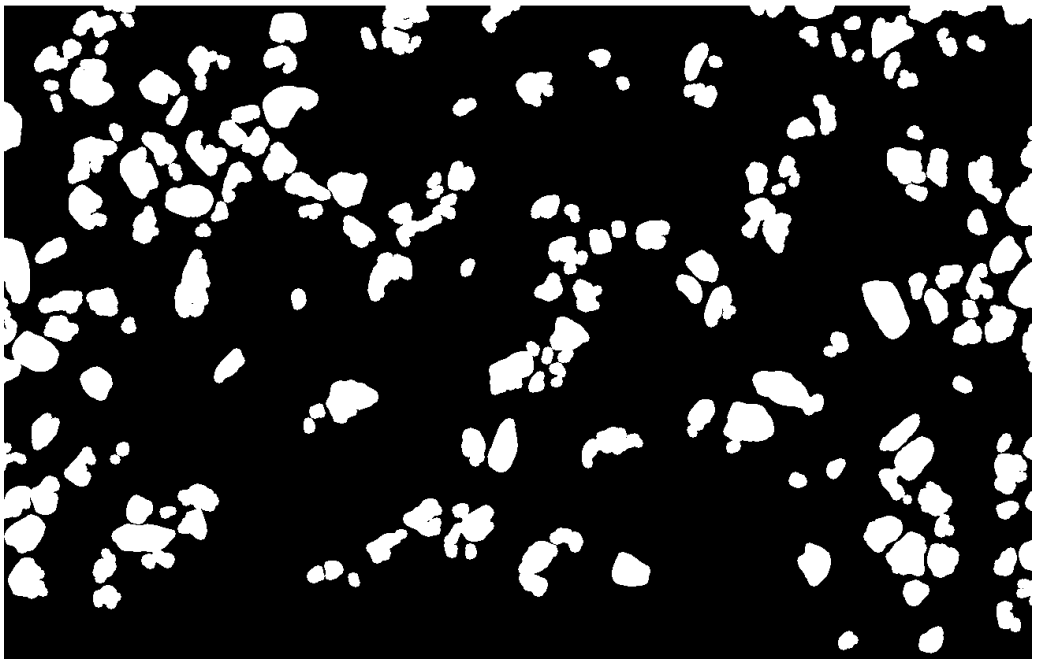
```
% La mida min i max definit per usuari
midaMAX = 3000;
midaMin = 200;

% Eliminar els components amb un nombre de píxels que estiguin fora del rang especificat
[BW, count]= RangPedretes(BW, midaMin, midaMAX);
imshow(BW);
```

La función RangPedretes, recibe por parámetros la imagen binarizada y el tamaño mínimo y máximo deseado de piedras. Con esa información, dicha funcionalidad utiliza la función bwconncomp para obtener los componentes conectados en la imagen binaria y, a continuación, filtra los componentes que no cumplen con el rango de píxeles especificado utilizando la función find y la comparación (np >= midaMin) & (np <= midaMAX), donde np es un vector que contiene el número de píxeles de cada componente conectado.

```
C = bwconncomp(BW);
% Creem una copia de BW on només quedaran les pedres que cumplin les
% condicions
CBW = false(size(BW));
% Obtenir el nombre de píxels de cada component
np = cellfun(@numel, C.PixelIdxList);
% trobem les pedres que cumpleixen les condicions
idx = find((np >= midaMin) & (np <= midaMAX));
```

La función luego crea una copia de la imagen binaria (CBW) en la que solo se incluyen los componentes que cumplen con las condiciones. Esto se logra estableciendo en verdadero los píxeles correspondientes a los componentes en la imagen CBW. La función devuelve la imagen binaria CBW como resultado, que contiene sólo los componentes que cumplen con las condiciones, y también devuelve la cantidad de componentes del resultado obtenido para poder mostrarlo por pantalla.

```
% preparem la copia de BW amb les pedres obtingudes en idx
for i = 1:numel(idx)
    CBW(C.PixelIdxList{idx(i)}) = true;
end
%montage({BW,CBW});
Res = CBW;
count = size(idx,2);
```



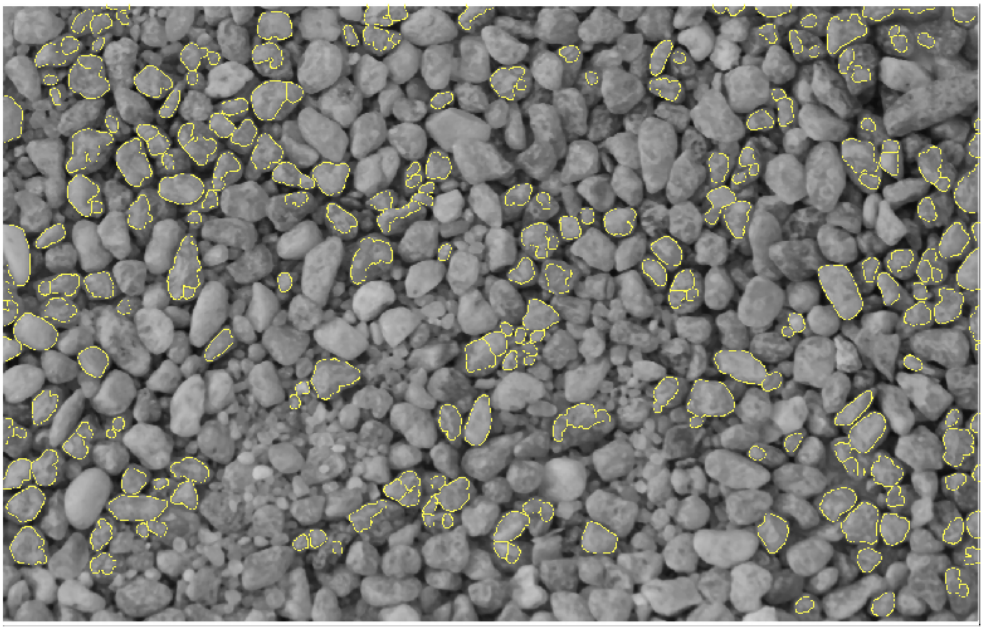
Una vez obtenida la imagen binarizada sin las piedras que no cumplen las dimensiones, procedemos a calcular su transformada de distancia. Le reducimos el ruido que pueda haber con un filtro de mediana, y finalmente le establecemos -Inf en los píxeles que no pertenecen al primer plano de la imagen para así poder separar posibles aguas.

```
% Obtenir la transformada de distància de la imatge binària
TD = -bwdist(not(BW), "quasi-euclidean");

% Aplicar un filtre de mitjana a la transformada de distància
TD = medfilt2(TD, [8 8]);
TD(not(BW)) = - Inf;
```

Ahora ya podemos segmentar la imagen obtenida con la función watershed. Así pues, obtenemos los contornos de las piedras resultantes para así poder, mediante imoverlay(), superponer los contornos sobre la imagen original en escala de grises.

```
WS = watershed(TD);

% Obtenir els contorns de les pedres resultants
IB = WS == 0;
RGB = imoverlay(I, IB);
imshow(RGB);

disp(['El número de piedras que cumplen las condiciones es: ', num2str(count)]);
```

Para acabar, gracias al método RangPedretes, podemos mostrar por pantalla la cantidad de piedras que cumplen las condiciones definidas.



```
El número de piedras que cumplen las condiciones es: 160
```

Anexo

```
% Obtenir els components connectats de la imatge binària
function [Res,count] = RangPedretes(BW, midaMin, midaMAX)
    C = bwconncomp(BW);
    % Creem una copia de BW on només quedaran les pedres que cumplin les
    % condicions
    CBW = false(size(BW));
    % Obtenir el nombre de píxels de cada component
    np = cellfun(@numel, C.PixelIdxList);
    % trobem les pedres que cumpleixen les condicions
    idx = find((np >= midaMin) & (np <= midaMAX));
    % preparem la copia de BW amb les pedres obtingudes en idx
    for i = 1:numel(idx)
        CBW(C.PixelIdxList{idx(i)}) = true;
    end
    %montage({BW,CBW});
    Res = CBW;
    count = size(idx,2);
end
```

codigo1:RangPedretes.m

```
I = rgb2gray(imread('imagen.jpg'));
imshow(I)

I = medfilt2(I,[4,4]); %reduir soroll

% Binarització de la imatge
level = 255*graythresh(I);
BW = I > level;
imshow(BW);

BW = imopen(BW, strel("sphere", 5));
BW = imdilate(BW, ones(2,2));
imshow(BW)

% La mida min i max definit per usuari
midaMAX = 3000;
midaMin = 200;

% Eliminar els components amb un nombre de píxels que estiguin fora del rang especificat
[BW, count]= RangPedretes(BW, midaMin, midaMAX);
imshow(BW);

% Obtenir la transformada de distància de la imatge binària
TD = -bwdist(not(BW), "quasi-euclidean");

% Aplicar un filtre de mitjana a la transformada de distància
TD = medfilt2(TD, [8 8]);
TD(not(BW)) = - Inf;
WS = watershed(TD);

% Obtenir els contorns de les pedres resultants
IB = WS == 0;
RGB = imoverlay(I, IB);
imshow(RGB);

disp(['El número de piedras que cumplen las condiciones es: ', num2str(count)]);
```

código principal:H4.mlx