Traitement et Synthèse d'Image

TP2 – Morphologie Mathématique

Le but de ce TP est de réaliser l'analyse granulométrique d'une image, i.e. de déterminer le nombre d'éléments composant l'image en fonction de leur taille. Il est donc nécessaire dans un premier temps de segmenter l'image en deux zones : les éléments d'intérêt, et le fond. Ensuite, les opérateurs de morphologie mathématique nous permettrons de déterminer le nombre d'éléments d'une taille et d'une forme prescrites, pour en déduire les courbes granulométriques.

Table des matières

1 Préparation

Implémenter l'algorithme des K-means à plusieurs régions (TP1, exercice 4).

2 Transformations d'histogramme

Cette première partie a pour but de segmenter l'image en deux régions distinctes, que sont le fond et les éléments d'intérêt.

1 – Lire et afficher l'image 'pieces.png'.
2 – Segmenter l'image en 2 régions on utilisant la fonction K-means implémentée lors du précédent TP. Commenter.
En utilisant la fonction K-means (cf annexe 1) pour avoir deux régions, on obtient la figure suivante :

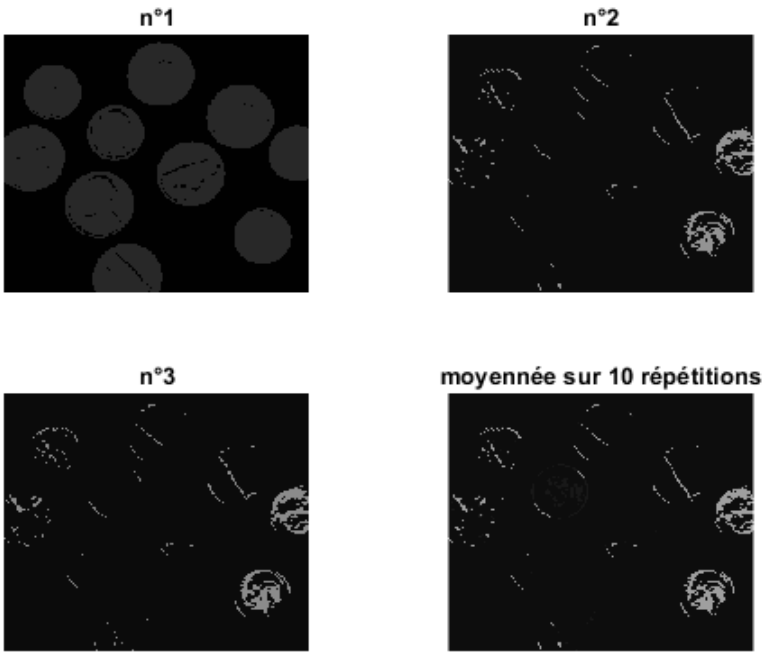


FIGURE 1 – K-means pour avoir deux régions

On constate que dans la mesure où la segmentation se fait à partir d'une valeur aléatoire associée à chaque région, les éléments d'intérêts, ici les pièces, ressortent mal du fond, voire être totalement mal identifiées. Une solution naïve serait de faire une moyenne de N K-means afin de diminuer les fluctuations statistiques. On utilise le code suivant :

```
n=10;
i_seg4 = k_means(image,2);
for i=1:n;
i_seg4 = i_seg4 + k_means(image,2);
end
i_seg4=i_seg4/n;
```

On constate qu'on a un meilleur résultat, mais on n'a plus une image binaire, dans le sens avec un niveau bas et un niveau haut dans les couleurs. Par ailleurs, l'identification des pièces est perfectible.

3 – Calculer et afficher l'histogramme de l'image à l'aide des fonctions `imhist` et `bar`. Puis calculer et afficher l'histogramme normalisé et l'histogramme cumulé de l'image.
On utilise le code suivant :

```
hist=imhist(image);
[h,w]=size(image);
hist_norm = hist/(h*w); % calcul de l'histogramme normalise
```

On obtient la figure suivant :

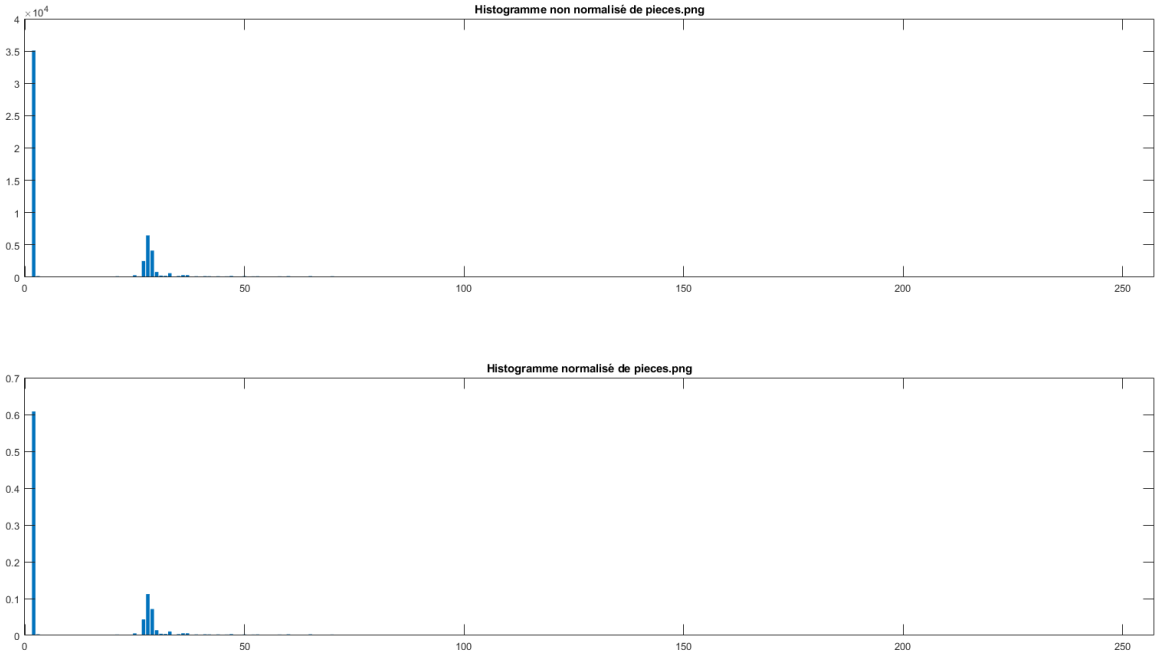


FIGURE 2 – Rendu des histogrammes

On constate que cette représentation normalisée peut être vue comme une répartition statistique de la couleur des pixels sur cette image.

Pour avoir l'histogramme cumulé, on utilise le code suivant :

```
hist_cumul = hist_norm;
cumul = 0;
for i = 1:length(hist_norm)
    cumul = cumul + hist_norm(i);
    hist_cumul(i) = cumul;
end
```

On obtient la figure suivante :

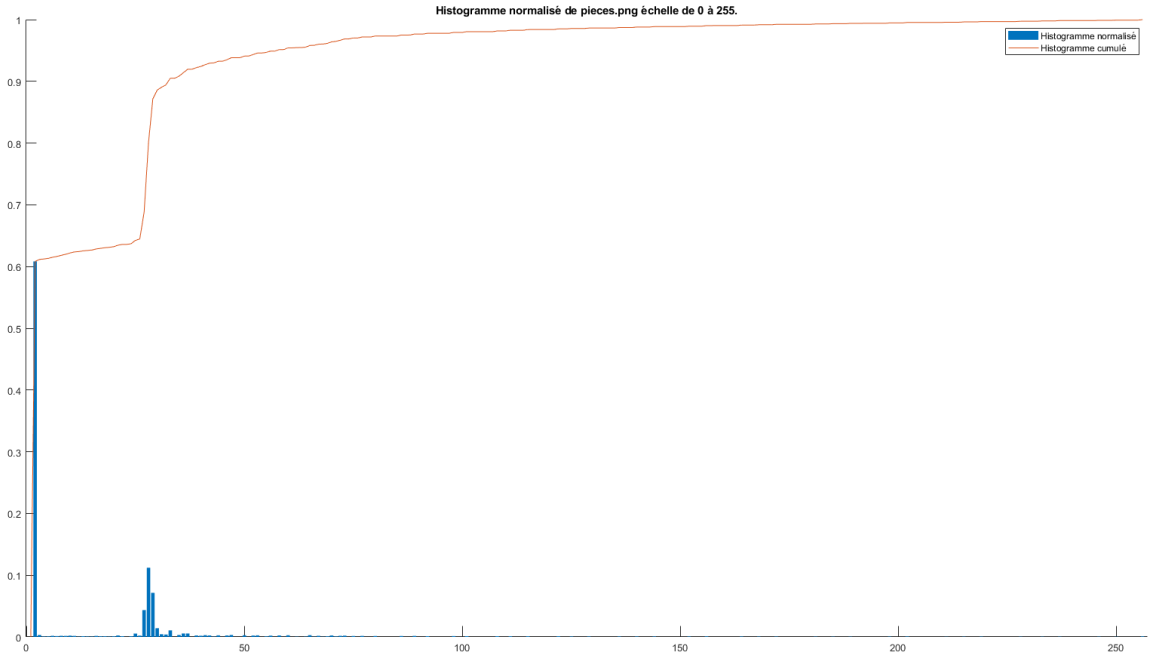


FIGURE 3 – Histogramme normalisé et cumulé

On constate qu'on a principalement des valeurs faibles, c'est-à-dire sombre et au-dessus de 50 on a 95% des valeurs, et au-dessus de 156 on a 99% des valeurs. On pourrait envisager de faire une dilatation pour étaler le spectre sur l'ensemble de la plage du niveau de gris : 0-255 en considérant ces valeurs. On aura une meilleure luminosité.
Par ailleurs, on constate qu'on a globalement deux regroupements, le premier pic à 2 contenant environ 60.83% des valeurs correspond au fond noir, majoritaire. Tandis que le regroupement autour de 28 correspond aux pièces.

4 – Implémenter l'égalisation d'histogramme pour cette image. On pourra éventuellement comparer le résultat avec la fonction Matlab correspondante.
5 – Calculer et afficher l'histogramme égalisé.
6 – Afficher l'image obtenue, son histogramme normalisé, et son histogramme cumulé sur une même figure. Commenter.
On utilise le code suivant :

```
[i_eg2,hist_eg2]=histeq(image); %solution matlab

% solution 1
% [h,w]=size(image);
% max=255;
% min=79;%solution identique de matlab
% i_eg1 = zeros(h,w);
%  for i=1:h*w;
%     i_eg1(i)=(max-min)/(h*w) * sum(hist(1:floor(image(i)*255)))+
    min;
```

```
%           g_k = (G-1)/N sum(i=1,i=k,H_i) cas non normalise
%  end
% i_eg1=i_eg1/255;

%solution2
[h,w]=size(image);
i_eg1 = zeros(h,w);
max=255/255;
min=0/255;
for i=1:(h*w)
    i_eg1(i)=((max-min)* hist_cumul(image(i)*255) + min+1);%image
       en flotant
        %cas normalise
end

hist_ceg1=imhist(i_eg1)/(h*w);
cumul = 0;
for i = 1:length(hist_ceg1)
    cumul = cumul + hist_ceg1(i);
    hist_ceg1(i) = cumul;
end
```

On obtient la figure suivante :

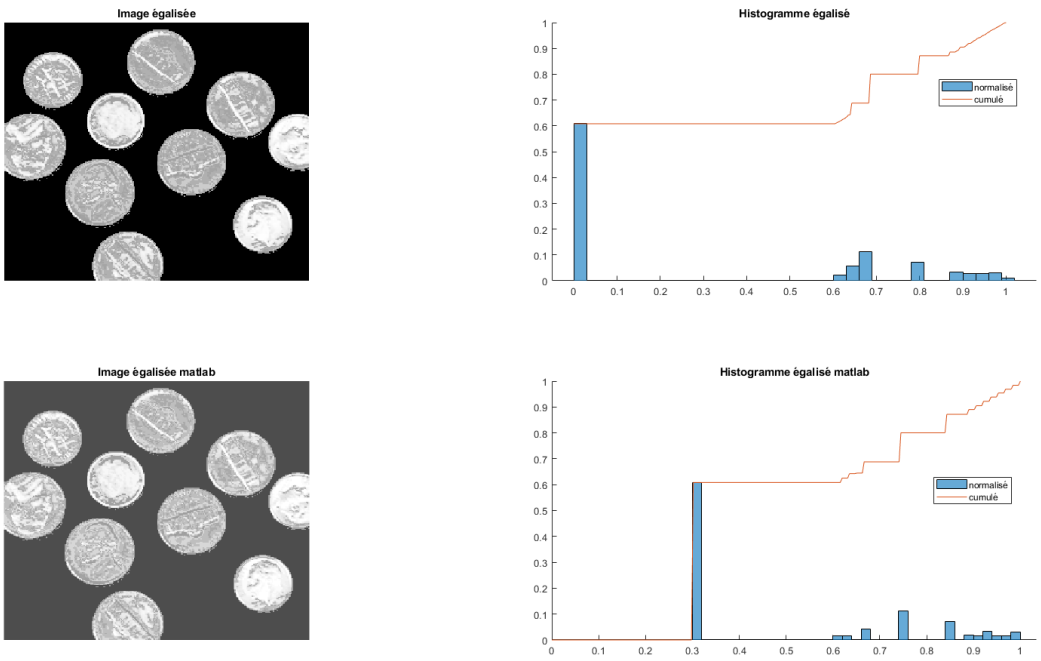


FIGURE 4 – Égalisation de l'histogramme.

On a fait le choix de conserver la deuxième solution, car celle-ci sera meilleure pour l'opération qui va suivre, cependant la première solution est ce qu'on attend. On a tout de même une égalisation, même si elle est imparfaite au début.
Par ailleurs on constate que la solution matlab calcul l'histogramme cumulé en faisant l'écart à

la courbe d'équation y=x. En effet, cela est visible par la courbe cumulée tracée nativement par matlab, qu'on a ici remplacé par notre solution.

7 – Segmenter l'image égalisée en 2 régions on utilisant de nouveau votre fonction K-means. Commenter.
On obtient la figure suivante :

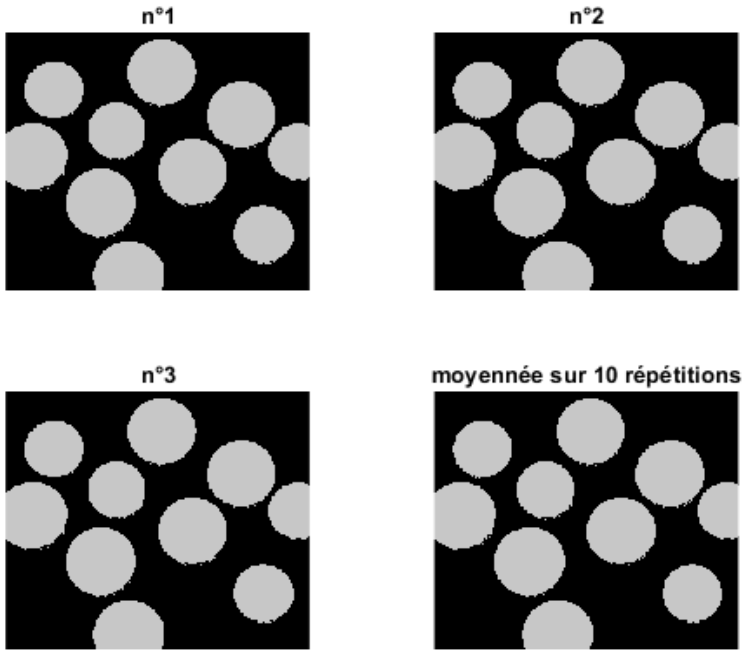


FIGURE 5 – K-means 2 régions sur une image égalisée

On constrate qu'on a un meilleur détourage que précédemment. Les pièces se détachent bien du fond. En revanche on n'a pas nécessairement une convergence des figures, car elles dépendent des valeurs des moyennes initiales fixées aléatoirement. On constate qu'il faut entre 2 à 7 répétitions afin d'avoir une convergence pour une figure donnée, et ce nombre augmente avec le nombre de segmentation qu'on souhaite.
Il est important de noter qu'on peut avoir des segmentations vides dans certains cas, ici on a un plafonnement autour de 20.

3 Morphologie mathématique : granulométrie

Dans cette partie on cherche à obtenir la courbe de granulométrie des pièces. Avant cela, pour ne pas fausser la statistique, on procède à quelques prétraitements afin de "nettoyer" l'image.

1 – Sur l'image binaire obtenue dans la partie précédente, les pièces ne sont pas tout à fait homogènes. Quelle opération de morphologie mathématique permettrait de rendre les pièces "homogènes" ? L'appliquer à l'image binaire précédemment obtenue.
On peut penser à faire une fermeture afin de combler les tous au sein des pièces. Mais il s'avère

que si nous faisons ça, nous pouvons avoir des aspérités sur les contours des pièces à cause des artefacts les entourant. Ainsi on privilégiera les ouvertures pour enlever ces artefacts et par la suite on peut faire des fermetures pour combler les trous au sein des pièces.
On a crée la fonction suivante pour facilité notre travail tout le long du TP (cf annexe 2)

```
On utilise alors le code suivant :\\
i_g1 = im2bw(i_segeg4);%permet d'avoir une image binaire
disp("Avant traitement");
disp(bweuler(i_g1));
%i_g=masque(i_g1,'fermeture','rond');
%i_g=masque(i_g1,'fermeture','croix');%laisse passer des anomalies
%i_g2=masque(i_g1,'ouverture','carree');%fermeture donne des
    resultats correctes
i_g2=masque(i_g1,'ouverture','d10'); %resultat meilleurs
```

On constate qu'on a les meilleurs avec d10, les autres ayant un résultat tout à fait acceptable, mais perfectible (cf figure 6 Traitement 2). Dans la mesure où cette partie repose principalement sur la morphologie, on se permet d'utiliser la fonction im2bw native de matlab qui renvoie une image binaire à partir d'un seuil choisi de manière optimale. Dans la mesure où ce n'est pas le but premier de ce TP, on n'a pas explicité ce seuil de détection, mais on aurait pu imaginer récupérer les valeurs des zones correspondantes aux deux zones et par la suite mettre une à 0 et l'autre à 255.
Cela revient à utiliser l'algorithme suivant développé en TP1 auquel on rajoute la position des coordonnées des pixels en question afin de faciliter leurs remplacements. On rappelle :

```
[h,w]=size(image);%image segmente en zone, ici deux
V=[,];
for i=1:h*w;
    if ismember(image(i),V);
        continue;
    else
        V=[V,image(i)];%renvoie deux valeurs
    end
end
```

On n'a pas dévelopé cet algorithme pour éviter des erreurs de type class notamment pour la fonction masque (annexe 2).

2 – Certaines pièces étant coupées par le champ de vue, on risque de mal estimer leur taille. On va donc supprimer les objets touchant le bord par reconstruction par marqueur. Quel marqueur définir pour cette reconstruction ? Justifier.
On doit définir un marqueur bord (cf annexe 2 se=se_bord).
Ce marqueur correspond au bord de l'image, on peut alors appliquer un algorithme de reconstruction pour détecter les éléments intersectants avec le bord. (cf figure 6 prétraitement 3). Par la suite on fait une soustraction de ces éléments déterminés avec l'image qu'on veut traiter. (cf figure 6 Traitement 3) Il nous reste alors plus que les éléments ne touchant pas le bord.

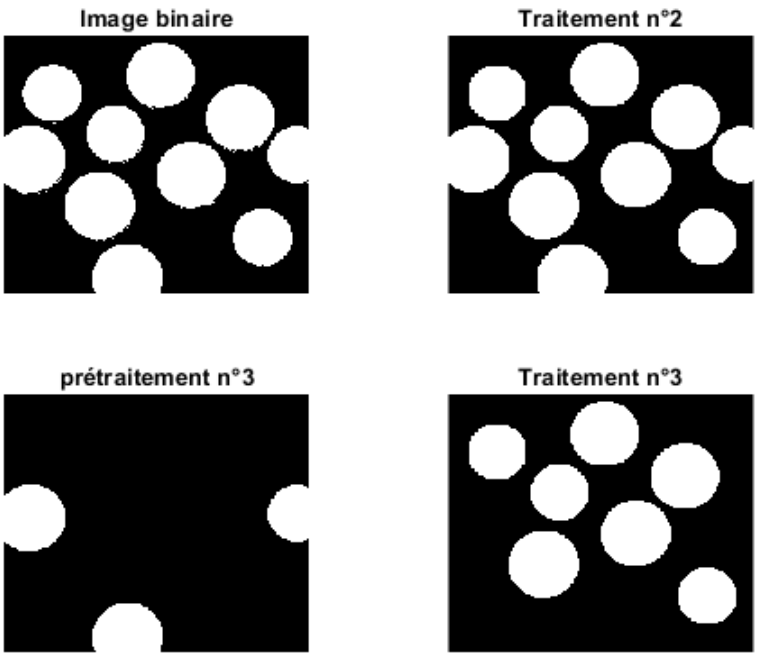


FIGURE 6 – Traitement de l'image

On doit faire ce nettoyage dans la mesure où on cherche par la suite à déterminer le rayon des disques. Or les disques intersectants le bord sont susceptibles de fausser notre analyse.

3 – Effectuer la reconstruction par marqueur pour éliminer les pièces touchant le bord. Pour cela, vous pouvez :

- utiliser la fonction matlab permettant de supprimer les objets du bord,
- *(Bonus) :* écrire vous-même l'algorithme à l'aide de dilatations, intersections (on affichera, à chaque itération, l'image de la reconstruction).

On utilise le code suivant :

```
i_gsb1 = masque(i_g2,'reconstruction','bord');%detecte les bords;
i_gsb2 = im2bw(i_g2-i_gsb1);%image sans element au bord
%i_gsb1 = imclearborder(i_g2);%equivalent matlab
```

On effecture la reconstruction pas à pas (cf annexe 3 mode="reconstruction"), nous faisons le choix d'afficher seulement certaines images ici. L'intégralité est en annexe 4.

FIGURE 7 – Reconstruction 4 , 12, 20 et 28 des éléments du bords

On remarque qu'on peut obtenir le même résultat plus rapidement si on prend l'élément structurant reconstructeur de manière plus optimale. On fera attention au fait que si l'élément struc-

turant est trop grand, on risque d'intersecter des éléments proches du bords mais qui ne l'intersectent pas.

4 – On souhaite obtenir les courbes de granulométrie. Pour cela, on utilisera notamment la fonction `bweuler`, et on étudiera la réponse de l'image nettoyée à une opération morphologique par un élément structurant correspondant à la taille et à la forme des objets considérés.

5 – Vérifier que les courbes obtenues sont cohérentes avec le contenu de l'image. Commenter.

On utilise le code matlab suivant pour déterminer le nombre de pièces ainsi que leurs tailles :

```
[centers, rayons, metric] = imfindcircles(i_gsb2,[5 200]);
nb_cercle = length(centers);
disp(metric);
viscircles(centers, rayons,'EdgeColor','b');%permet d'afficher sur
    l'image les cercles
```

On obtient la figure suivant :

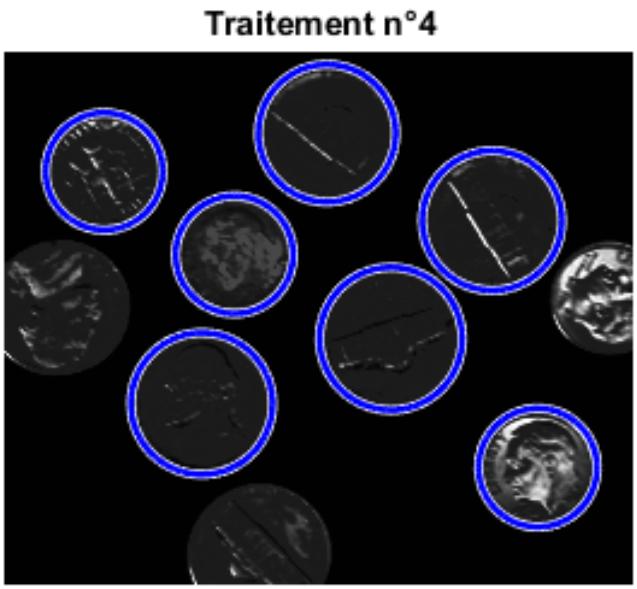


FIGURE 8 – Identification des pièces par matlab

Avec la fonction matlab, on a 7 pièces de tailles suivantes :
29.6089633119596
29.3954054657509
28.9782446181189
28.5101834730687
24.5963238583304
24.4027966476294
24.3171341168958

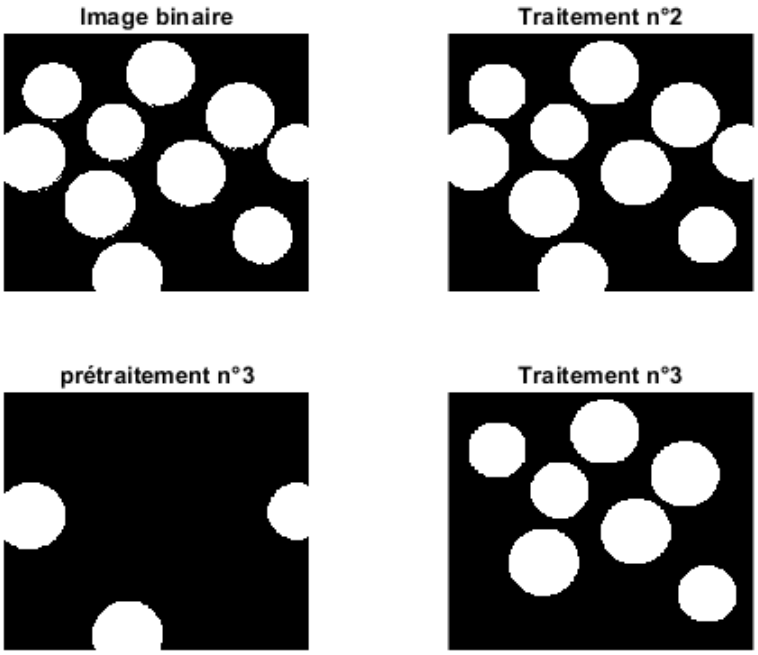
On développe notre propre algorithme (annexe 3). Cet algorithme repose le fait de conserver

l'étape avant une érosion totale menant à la première disparition de l'image. Par la suite on procède à l'étape contraire et on reconstruit l'image à partir de ce point. On soustrait cette reconstruction, qu'on prendra soin d'être légèrement plus grande afin de s'assurer du fait de bien enlever l'élément en question. On réitère, ainsi de suite, en enlevant les pièces une part une.
On a la figure suivant :

FIGURE 9 – Détermination différente famille de pièces

On constate ainsi qu'on enlève bien les pièces les plus grosses jusqu'aux plus petites. Cette méthode a également l'avantage de regrouper les pièces par tailles semblables de prime abord. En détail on constate qu'on a une taille à l'unité près ce qui provoque une scission du groupe de 4 pièces les plus grandes. Avec notre algorithme on a :
29
28
24
On constrate que notre résolution étant trop petite, on a un regroupement par paquet. Par ailleurs, en s'appuyant sur la figure précédente on peut rétablir le nombre, et regroupe à nouveau les pièces : 4 grandes de taille 28, et 3 petites de tailles 24, à plus ou moins un près. Une solution pour améliorer notre résultat et pas, serait de travailler potentiellement sur un élément structurant plus fin. Cependant on peut de manière volontaire regrouper les pièces de même taille au sein d'un même paquet, dans ce cas-là, on augmentera plutôt la taille de l'élément structurant.
On remarque par ailleurs qu'on aurait pu prendre deux éléments structurants différents pour l'érosion et pour la dilatation.

On remarque qu'en utilisant la fonction bweuler sur les images précédentes :



On a pour l'image binaire la détection de 18 éléments, après passe d'un élément structurant par fermeture on a 10 éléments, car comprenant les 3 pièces bords. Puis pour le traitement 3 on a bien 7 éléments. Cela montre donc la pertinance des traitements que nous avons fait précédemment pour déterminer le nombre de pièce, traitement jusqu'à 2, et pour déterminer la taille des différentes pièces (traitement 3 et plus).

4 Annexes

4.1 Fonction K-means

On a le code suivant pour la fonction qui prend en entrée une image avec le nombre de segmentation qu'on souhaite. La fonction retour l'image avec le nombre de région correspondant.

```
function [matrice,moyennes]=k_means(image,k)
        %K-means
        k=ceil(k);
        retour=zeros(size(image));%image vide
        %k nombre de zone
        nn=0;
        if k>1%avoir plus d'une zone
                A=sort(rand(k,1));
                epsi=1/255;% pixel pres
                j=2;
                while j>epsi;
                        nn=nn+1;
                        Av=A;
                        B=0*(1:k-1);
                        for i=1:(k-1);%Moyenne des termes
                                B(i)=(A(i)+A(i+1))/2;
                        end
                        %Valeurs extremums
                        A(1)=mean(mean(image(find(image<=B(1)))));
                        A(k)=mean(mean(image(find(image>B(k-1)))));
                        for i=1:(k-2)
                                A(i+1)=mean(mean(image(find((image >
                                    B(i) & image<B(i+1))))));
                        end
                        j=max(abs(Av-A));
                end
                retour = retour + A(1)*(image<=B(1));
                retour = retour + A(k)*(image>B(k-1));

                for i=1:(k-2)
                        retour = retour + A(i+1)*(image > B(i) &
                           image<B(i+1));
                end
        end
        disp(nn);
        matrice=retour;
end
```

4.2 Fonction masque

On a le code suivant pour la fonction permettant de faire de la morphologie. On a en entrée l'image avec différent mode :
- érosion
- dilation
- ouverture
- fermeture
- reconstruction

avec différents éléments structurants :
- croix
- carrée
- rond
- disque de taille 10

On nous renvoie une image.

```
function matrice=masque(image,mode,forme)
        [h,w]=size(image);
        %element structurant
        se=[1];%par defaut
        se_croix = [0,1,0;1,1,1;0,1,0];
        se_car = [1,1,1;1,1,1;1,1,1];
        se_rond = [0,0,1,0,0;
                           0,1,1,1,0;
                           1,1,1,1,1;
                           0,1,1,1,0;
                           0,0,1,0,0];
        se_d10 = strel('disk', 10);
        se_bord = zeros(h,w);
                se_bord(:,1)=1;
                se_bord(1,:)=1;
                se_bord(:,w)=1;
                se_bord(h,:)=1;

        % selectionne l'element structurant
        if strcmp(forme,'croix');
                se=se_croix;end
        if strcmp(forme,'carree');
                se=se_car;end
        if strcmp(forme,'rond')
                se=se_rond;end
        if strcmp(forme,'d10')
                se=se_d10;end
        if strcmp(forme,'bord')
                se=se_bord;end

        retour = zeros(h,w);
        %fait l'operation
        if strcmp(mode,'dilatation');
                retour = imdilate(image,se) ;end
        if strcmp(mode,'erosion');
```

```
36                  retour = imerode(image,se);end
37          if strcmp(mode,'ouverture');
38                  retour = imerode(image,se);
39                  retour = imdilate(retour,se) ;end
40          if strcmp(mode,'fermeture');
41                  retour = imdilate(image,se) ;
42                  retour = imerode(retour,se);end
43
44          if strcmp(mode,'reconstruction');
45                 %figure(40);
46                  imshow(image);
47
48                  marker = im2bw(se);%meme classe d'objet
49                  image=im2bw(image);
50
51                  retour = marker & image;
52                  for i = 1:100;
53                          retour_av=retour;
54                          retour = imdilate( retour & image, se_rond )
                                & image ;%peut etre plus rapide si on
                               choisit un autre masque
55                          if retour==retour_av;%si les bords sont
                               entieremements detectes
56                                  break
57                          end
58                  end
59                  %retour = imreconstruct(marker, image);%equivalent
                       matlab
60          end
61
62          matrice = retour;
63  end
```

4.3 Fonction compteur

On a le code suivant pour la fonction compteur qui prend en entrée une image, de préférence binaire et qui renvoie en sortie une liste du nombre de rayons différents à la taille d'entier près.

```
function [rayon]=compteur(image);
        nb_f=20;
        sav=im2bw(image);
        retour=im2bw(image);
        r=[,];
        [h,w]=size(image);
        vide=im2bw(zeros(h,w));

        while 1==1;
                disp('Traitement en cours');
                image_r=im2bw(sav);

                for i=1:100;
                se = strel('disk',i);
                image_rav=image_r;
                image_r=imerode(sav,se);
                        if image_r==image_rav;
                                disp('Dimension trouvee');
                                r=[r,i];
                                break
                        end
        %     end

                %matrice=retour;
                retour=imdilate(image_rav,strel('disk',i+3));
                end
                sav=im2bw(sav)-im2bw(retour);
                if sav==vide;
                        break;
                end
                nb_f=nb_f+1;
                figure(nb_f);
                imshow(retour);
                %figure(nb_f+1);%affiche etape ou on soustrait
                %imshow(sav);
        end

        disp('Fin');
        rayon = r;
end
```

4.4 Reconstruction complète pas à pas