IN1 : Traitement d'image

Compte rendu de TP

DELZERS Rémi M1 SIS

Table des matières :

I. Présentation du code

Fonction de lissage :

```
function [t] = lissage(nom_fichier)
debut=cputime;


I=imread(nom_fichier,'jpg');% Lecture en Jpeg
seuil=0.01;%dÃ©finition du seuil

% Reduction de la taille d'une image :
while (((size(I,1))*(size(I,2))) > 16384 )
    I2=I(1:2:end,1:2:end,:);
    I=I2;
end

% Selection de la version de prelissage :
%   - 1ere version : 3 couleurs avec max
%   - 2e version : 3 couleurs avec min
%   - 3e version :  8 couleurs
%   - 4e version : utilisation d'histogrammes pour determiner les seuils de
%   couleur et de region

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

% 1ere version :
%[I,r]=prelissage1(I);
%%%%%%%%%%%%%%%%%%%%%%

% 2e version :
 %[I,r]=prelissage2(I);
%%%%%%%%%%%%%%%%%%%%%%

% 3e version :
% [I,r]=prelissage3(I);
%%%%%%%%%%%%%%%%%%%%%%

% 4e version
%[I,r]=prelissage4(I);
% Calcul du seuil idï¿½al ï¿½ l'aide d'un histogramme
%for i=1:max(size(r))
%    S(i)=Surface_region(I,r(i))/(prod(size(I)));
%end
%N=hist(S,20);
%[val,i]=max(N);
%s=sum(N);
%t=val/s;
%seuil=(t*i*5)/100;

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


[rp,S]=region_ng(I,r,seuil); %Calcul des regions negligeables

for i=1:length(rp)
    I(find(I==rp(i)))=Region_incluante(rp(i),I,r);%fusion des regions negligeables a leur region
%incluante
end
    figure
    imagesc(I); %affichage de l'image segmentée
fin=cputime;
t=fin-debut;
```

Fonction prelissage1 :

```
function[I,region]=prelissage1(I)
%nom_fichier est le nom de l'image
%I est l'image avec rï¿½gions numï¿½rotï¿½es de 1 ï¿½ n.
%region est le vecteur 1:n
%Lecture et affichage de l'image avec extraction des rï¿½gions (de 1 ï¿½ 3)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[I2 I]= max(I,[],3); %renvoie une matrice de valeur dans I2 et une matrice d'index dans I
image(I.*10); %accentue la luminosite
colormap gray %met en niveau de gris

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Re-numï¿½rotation des rï¿½gions de 1 ï¿½ n, chaque rï¿½gion sera numï¿½rotï¿½e de 1 ï¿½
%n.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Ip=I;
a=4;
while sum(sum(Ip))~=0
    var=0;
    i=1;
    j=1;

    % On cherche un pixel d'une rï¿½gion non nulle
    while var==0
        if Ip(i,j)~=0
            indj=j;
            indi=i;
            var=1;
        end
        i=i+1;
        if i>size(I,1)
            j=j+1;
            i=1;
        end
    end

    %On applique l'algo FloodFill
    pixels=[];
    pixels=FloodFill(Ip, indi, indj, Ip(indi,indj), pixels);

    % Les pixels (de la rï¿½gion) rendus par FloodFill sont mis ï¿½ zero pour l'image
    % temporaire Ip et renumï¿½rotï¿½s sur I.
    for i=1:size(pixels,1)
        Ip(pixels(i,1), pixels(i,2))=0;
        I(pixels(i,1), pixels(i,2))=a;
    end

    a=a+1;
end %on obtient l'image renumerotee en regions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Sauvegarde et affichage de I
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
I=I-3;
save I I
region=1:max(max((I)));
figure
image(I);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% NE PAS S'ATTARDER SUR CETTE FONCTION
function pixels = FloodFill(S, ix, iy, seuil, pixels)
%2D flood-fill algorithm %
% Inputs:
%S = S(ix,iy) = 2D surface
%(ix,iy) = node indices
%pixels = list of pixels in the current stack (initially set as pixels = [])
%seuil = pixels value for which the filling is accomplished
% Outputs:
%pixels = list of pixels on the same stack %

[nx,ny] = size(S);
pile=[ix,iy];
```

```
while(isempty(pile)==0)

    x=pile(1,1);
    y=pile(1,2);

    if length(pile)<=1
        pile=[];
    else
        pile=pile(2:end,:);
    end
    if(S(x,y)==seuil)
        S(x,y)=0;
        pixels = [ pixels; [x y] ] ;
        if(x+1<=nx && S(x+1,y)==seuil)

            pile=[pile; x+1 y];end
        if(y+1<=ny && S(x,y+1)==seuil), pile=[pile; x y+1];
        end
        if(x>1 && S(x-1,y)==seuil), pile=[pile; x-1,y];end
        if(y>1 && S(x,y-1)==seuil), pile=[pile; x,y-1];end
    end

end
```

Changements apportés a prélissage 2 :

```
%Lecture et affichage de l'image avec extraction des rï¿½gions (de 1 ï¿½ 3)

[I2 I]= min(I,[],3); %renvoie une matrice de valeur dans I2 et une matrice d'index dans I
image(I.*10); %accentue la luminosite
colormap gray %met en niveau de gris
```

Changements apportés à prélissage 3 :

```
%Lecture et affichage de l'image avec extraction des rï¿½gions (de 1 a 8)

I1= (I(:,:,1)>127) + ((I(:,:,2)>127)*2) + ((I(:,:,3)>127)*4) +1;
I=I1;
image(I.*10)
```

Changements apportés à prélissage 4 :

```
function[I,region]=prelissage4(I)
%nom_fichier est le nom de l'image
%I est l'image avec rï¿½gions numï¿½rotï¿½es de 1 ï¿½ n.
%region est le vecteur 1:n


%Lecture et affichage de l'image avec extraction des rï¿½gions (de 1 a 8)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

N=hist(double(I(:,:,1)),[0:5:255]);
s=0;
x=1;
while(s<(size(I,1)*size(I,2))/2)
s=s+sum(N(x,:));
x=x+1;
end
R=(x*5)-5; %calcul du seuil idéal pour la composante rouge

N=hist(double(I(:,:,2)),[0:5:255]);
s=0;
x=1;
while(s<(size(I,1)*size(I,2))/2)
s=s+sum(N(x,:));
x=x+1;
end
G=(x*5)-5; %calcul du seuil idéal pour la composante verte


N=hist(double(I(:,:,3)),[0:5:255]);
s=0;
x=1;
while(s<(size(I,1)*size(I,2))/2)
s=s+sum(N(x,:));
x=x+1;
```

```
end
B=(x*5)-5; %calcul du seuil idéal pour la composante bleue


I1= (I(:,:,1)<R) + ((I(:,:,2)<G)*2) + ((I(:,:,3)<B)*4) +1;
I=I1;
image(I.*10)
```

Région négligeable :

```
function [rp,S] = region_ng(I,r,seuil)

rp=[];
for i=1:max(size(r))
    S(i)=Surface_region(I,r(i))/(prod(size(I)));
    if(S(i)<seuil)
        rp=[rp r(i)];
    end
end
```

Surface région :

```
function [S] = Surface_region(I,r)

S=sum(sum(I==r));
```

Pixel voisin :

```
function[R1,R2] = pixel_voisin(x,y,I)

R1=I(x+1,y); %calcule les deux voisins d'un pixel donnÃ© par leur coordonnÃ©es
R2=I(x,y+1);
```

Région incluante :

```
function [R_inc]=Region_incluante(ri,I,r)

temp=zeros(1,length(r));
for x=1:max(size(I,1))-1
    for y=1:max(size(I,2))-1
        if I(x,y)==ri
            [R1,R2]=Pixel_voisin(x,y,I);
            if R1~=ri
                temp(R1)=temp(R1)+1;
            end
            if R2~=ri
                temp(R2)=temp(R2)+1;
            end
        end
    end
end
[var,R_inc]=max(temp);
```

II. Commentaire de l'algorithme et améliorations

Principe de l'algorithme : on associe à une image une matrice de chiffres entre 1 et 3 représentant la composante de couleur d'un pixel ayant la valeur maximale. Une variante de l'algorithme consiste à prendre à la place la composante ayant la valeur minimale. On dresse à partir de cette matrice une liste de régions et on affecte à chaque pixel l'identifiant de région auquel il appartient. A partir de cette nouvelle matrice, on calcule les régions qu'on estime négligeable selon que leur surface est inférieure à un certain seuil, puis on les fusionne à leur région incluante. On obtient alors une matrice d'un nombre réduit de régions qu'on peut interpréter comme une image.

On constate que le maximum et le minimum ont tour à tour des meilleurs résultats selon les images : une amélioration pourrait consister en la normalisation de la luminosité des images pour supprimer ces variations.

On observe que la représentation par seulement 3 chiffres est insuffisante pour correctement représenter la couleur d'un pixel. On modifie alors l'algorithme pour qu'il divise chaque composante en deux espaces, un de faible intensité et un de haute intensité. Par la combinaison de chaque composante, on obtient alors un chiffre compris entre 1 et 8 pour représenter la couleur d'un pixel. Cela nous permet d'obtenir des régions mieux différenciées.

On constate qu'on obtient de meilleurs résultats en coupant la dynamique des couleurs à la moitié mais dans certains cas ou la plupart des pixels ont une couleur d'intensité équivalente on aimerait que l'algorithme s'adapte à la répartition des pixels dans les intensités de couleur.

Une solution qui pourrait être proposée serait l'utilisation d'un histogramme permettant de déterminer quelle est la valeur moyenne de chaque composante, c'est à dire qu'il y ai autant de pixels ayant une valeur inférieure que de pixels ayant une valeur supérieure à cette moyenne.

L'utilisation d'un histogramme nous permettrait également de déterminer un seuil idéal à partir duquel une image est considérée comme négligeable, en mesurant le pourcentage de régions de petite taille.

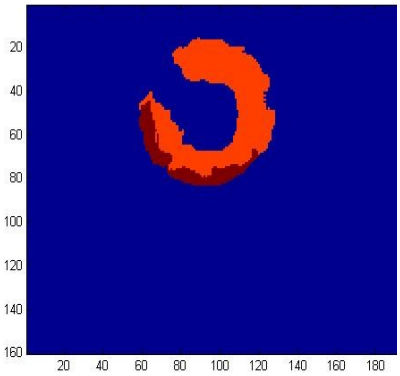
Nous chercherons ici à répondre à ces problématiques.

III. Résultats obtenus

On calculera le temps d'exécution t de chaque algorithme pour l'image 1.

Résultats de pré lissage 1 et 2 :

Image 1 : *Résultat avec max :* *Résultat avec min :*



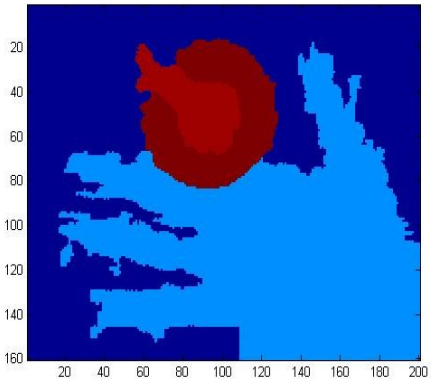
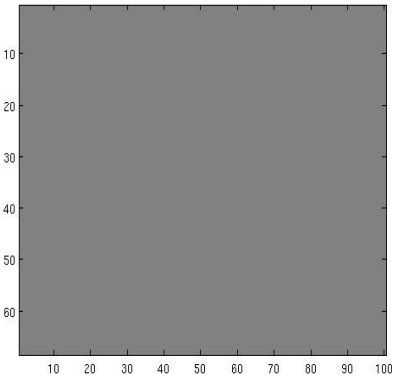


Image 2 :



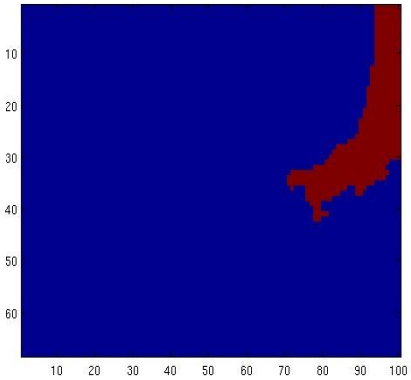
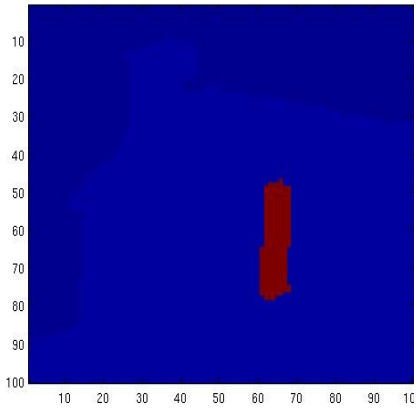


Image 3 :



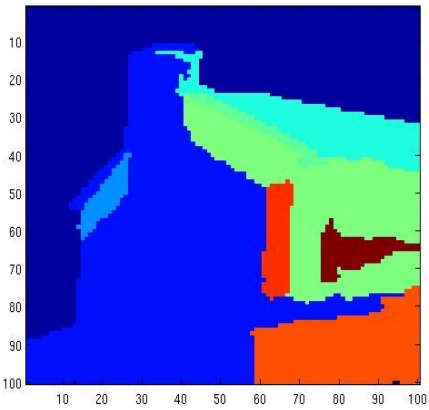
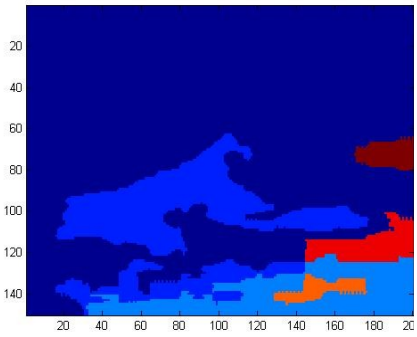


Image 4 :



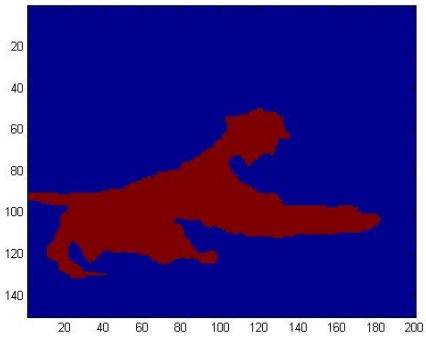
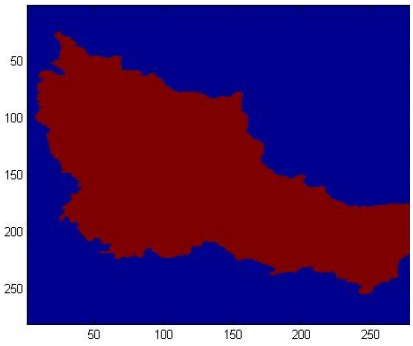


Image 5 :



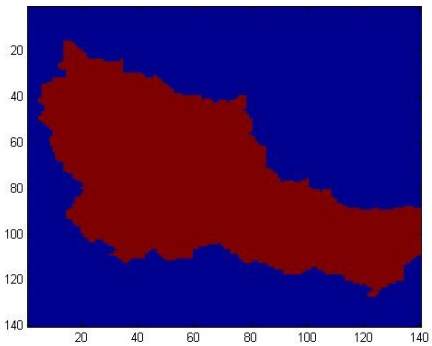
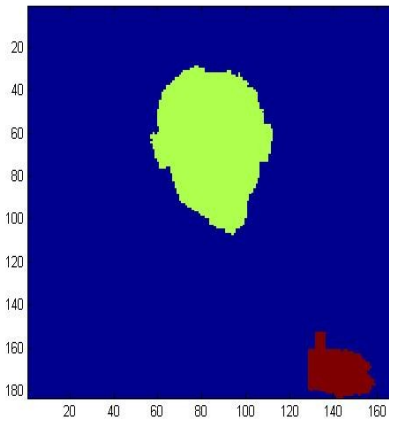
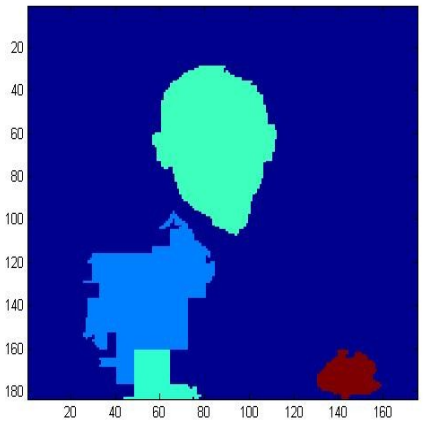


Image 6 :





t-max=0.4600

t-min=0.3300

On constate que le résultat est meilleur avec min ou max selon les images, on suppose d'un lien avec les propriétés de couleur des images.

Résultats de prélissage 3 :

Image 1:

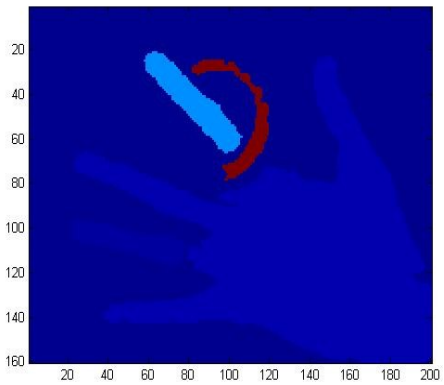


Image 2 :

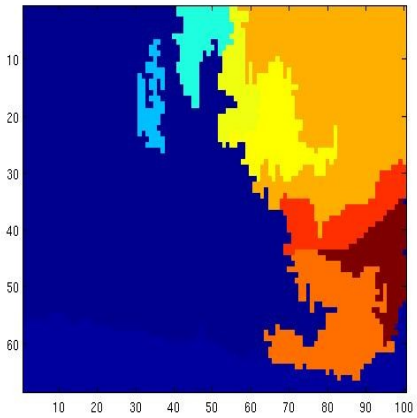


Image 3 :

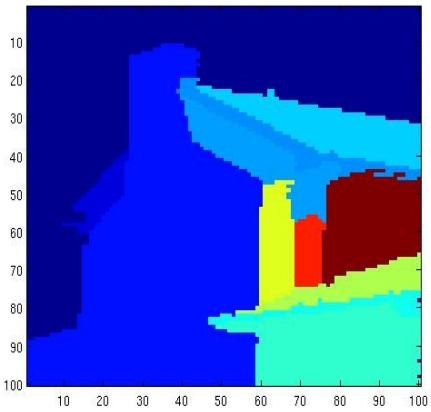


Image 4 :

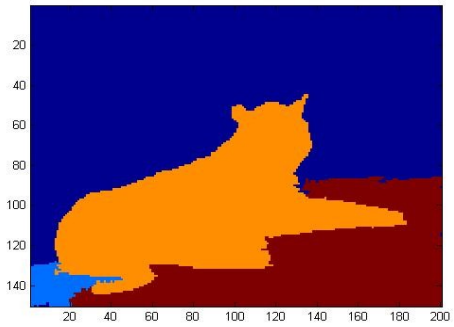


Image 5 :

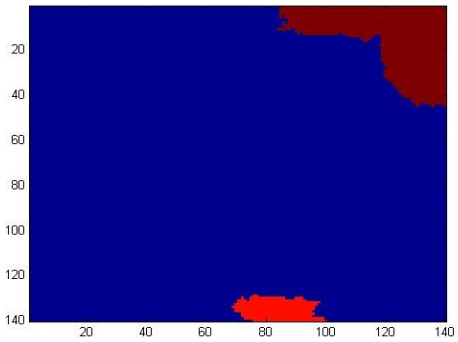
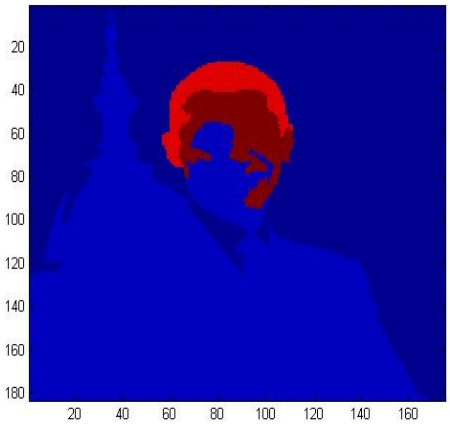


Image 6 :



t=0.4100

Cette version semble être une bonne amélioration car pour la plupart des images on obtient un résultat plus exploitable que pour la version min/max.

Résultats de prélissage 4 :

Image 1 :

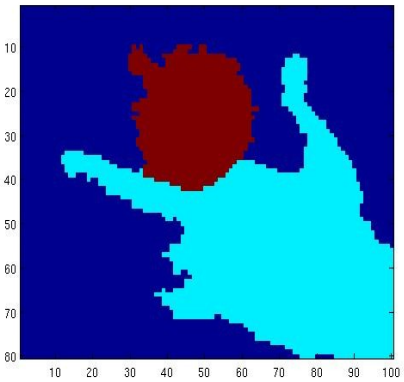


Image 2 :

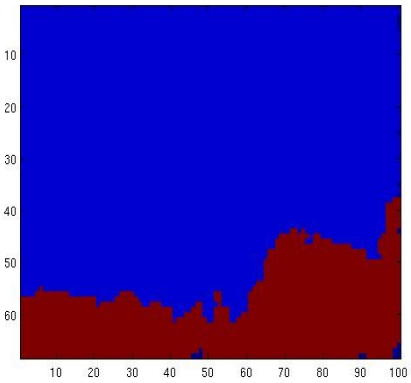


Image 3 :

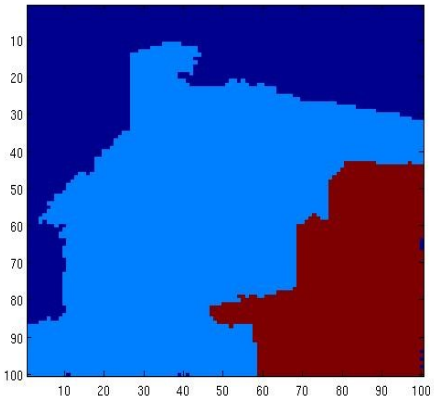


Image 4 :

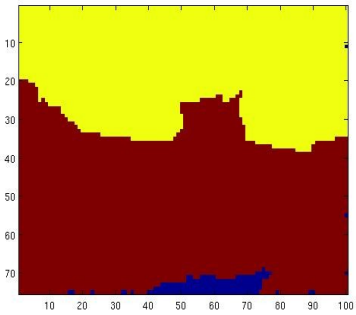


Image 5 :

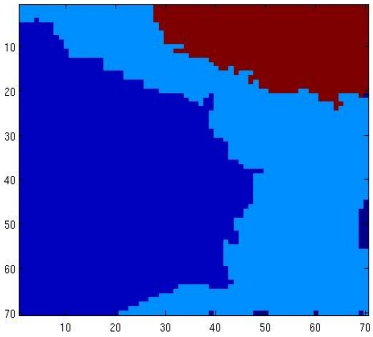
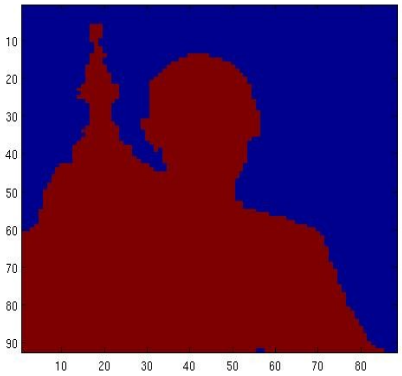


Image 6 :



t=0.8800

On constate que certains résultats ne sont pas vraiment satisfaisants. De plus leRapport-IN1 temps de calcul double comparé aux versions précèdentes. Cette version de l'algorithme n'est donc pas encore totalement au point.

IV. Comparaison avec JSEG

L'algorithme JSEG utilise le fait que l'on représente un pixel par un vecteur à 5 dimensions. La distance entre deux pixels correspond à leur distance euclidienne. On représente les k plus proches voisins d'un pixel p par les k pixels ayant la distance euclidienne à p la plus faible. L'algorithme JSEG construit une liste des k plus proches voisins de chaque pixel, puis construit des régions en construisant le couple des pixels les plus proches. On considère ensuite le centre de gravité de ce couple, et leur couleur moyenne. On calcule pour les n-2 pixels restants les voisins à la première région. On obtient les k plus proches voisins de cette région. On reproduit l'opération jusqu'à avoir pris tout les pixels. Au final on obtient trop de régions. Un paramètre de JSEG va fusionner les régions dont la distance est inférieure à un seuil.

Résultats de Segdist avec une merge de 0.3:

Image 1 :

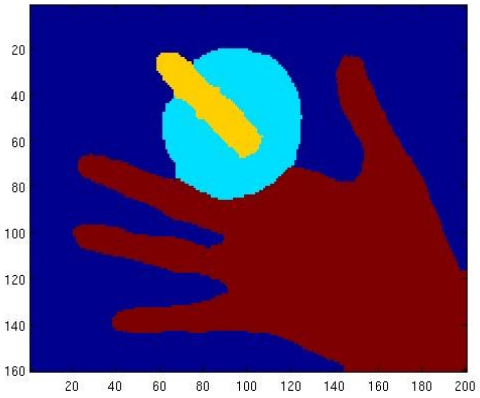


Image 2 :

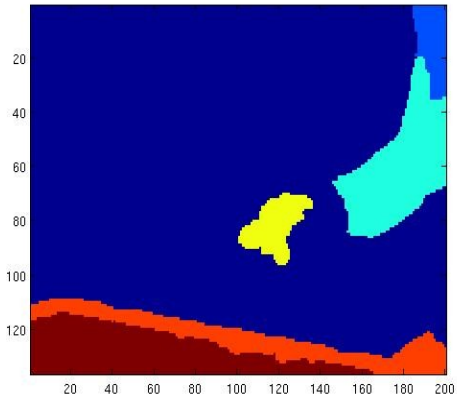


Image 3 :

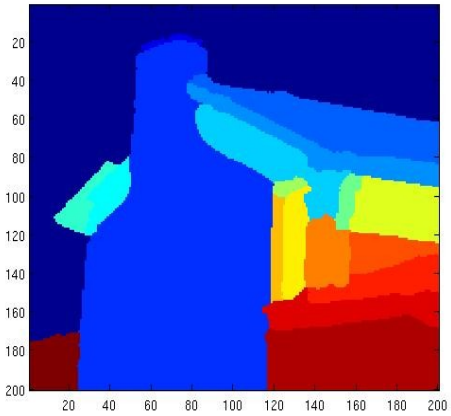


Image 4 :

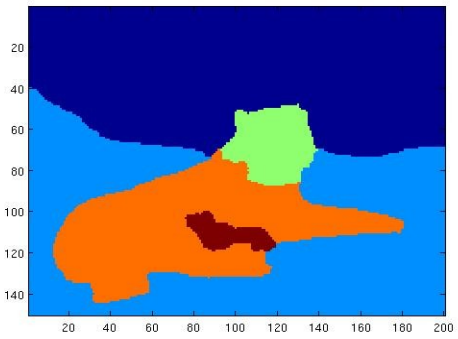


Image 5 :

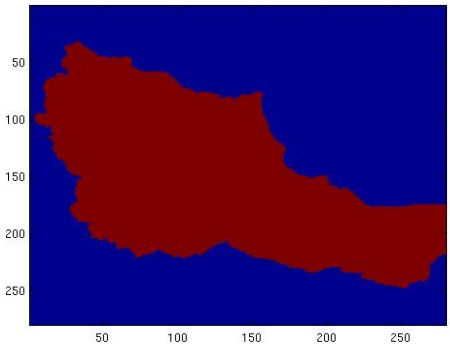
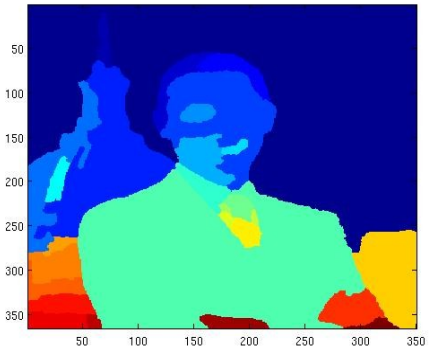


Image 6 :



Résultat de la commande : time segdist -i image1.jpg -t 6 -r9 seg1.gif -m 0.3

0.616u 0.000s 0:00.62 98.3% 0+0k 0+8io 0pf+0w

On peut en déduire que JSEG est un peu plus lent que les versions basiques de notre algorithme.

Conclusion : *On constate que certains résultats obtenus avec les différentes versions de notre algorithme basique s'approchent des résultats obtenus avec JSEG, même si ceux ci sont quand même plus satisfaisants. Il reste donc encore des améliorations à apporter à des algorithmes de segmentation d'image tels que JSEG.*