Davin
Kevin
M1 Info

Rapport Matlab

```
function[]=main(nom_fichier)

   [I,region]    = prelissage(nom_fichier);
   rp       = RegionNegligeable(I, region);

   while(length(rp) ~= 0)
      for i=1:length(rp)
I(find(I==rp(i))) = Region_incluante(rp(i), I, region); //Indexation des valeurs des région dans une matrice afin d'avoir un parallèle entre pixel d'une image et la région dans laquelle un pixel est incluse.
      end

      for i =1: length(rp)
         region(find(region == rp(i))) = []; //Supression des régions négligeable de la liste des régions.
      end

      rp       = RegionNegligeable(I, region);
   end

   figure
   imagesc(I)

function[rp]    = RegionNegligeable(I, r)
   rp=[];                     //Création d'une liste vide
   for i = 1 : length(r)      //Pour tous les pixels de l'image
      if (SurfaceRegion(I, r(i))/prod(size(I)))<0.01 //Si la surface d'une région représente moins de 1% de l'image global alors l'inscrire dans la liste des région petite (rp).
         rp    = [rp r(i)];
      end
   end

function[S]    = SurfaceRegion(I, r)
   S = sum(sum(I == r));  //Fonction qui rend la surface d'une région donné en argument.

function[I,region]=prelissage(nom_fichier)
%nom_fichier est le nom de l'image
%I est l'image avec régions numérotées de 1 à n.
%region est le vecteur 1:n


%Lecture et affichage de l'image avec extraction des régions (de 1 à 3)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
I=imread(nom_fichier,'jpg');% Lecture en Jpeg
%I=I(1:4:end,1:3:end,:);
figure
image(I)
%[I2 I]= max(I,[],3);
I    = (I(:, :, 1) < 128) + ( 4 * (I(:, :, 2) < 128) + 2 * (I(:, :, 3) < 128));
I    = I + 1;
figure
imagesc(I)
colormap gray
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%Re-numérotation des régions de 1 à n, chaque région sera numérotée de 1 à
%n.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Ip=I;
a=4;
while sum(sum(Ip))~=0
   var=0;
   i=1;
   j=1;

   % On cherche un pixel d'une région 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 région) rendus par FloodFill sont mis à zero pour l'image
   % temporaire Ip et renumérotés 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
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

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

% 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
```

```
function[R1, R2]    = PixelVoisin(x, y, I)  //Fonction qui récupère les valeur des pixels droit et bas d'un pixel donné en argument.
   R1    = I(x +1, y);
   R2    = I(x, y +1);
```

```
function [R_inc] = Region_incluante(Region_incluse, Image, Region)
      %Region incluse appartient à  RP;
// Fonction qui se charge pour une région donné, d'établir une liste des Régions dans TMP, qui seront augmenté  en fonctions des valeurs (de l'appartenance au région) d'un pixel, qui en fin de fonction ne gardera que la région qui sera la plus haute.
      TMP = zeros(1, length(Region));
      for x = 1:(size(Image, 1)) -1
         for y = 1:(size(Image,  2)) -1
            if (Image(x, y) == Region_incluse)
               [R1, R2] = PixelVoisin(x, y, Image);
               if (R1 ~= Region_incluse)
                  TMP(R1) = TMP(R1)+1;
               end
               if (R2~= Region_incluse)
```

```
            TMP(R2) = TMP(R2)+1;
          end
        end
    end
end
[var, R_inc] = max(TMP);
```

Discussion du code :

Si nous établissons l'image avec cette variable : [I2 I]= max(I,[],3);
Dans ce cas, nous nous privons d'information relative aux deux autres couleurs, ne gardant que le max d'une couleur au lieu de fonctionner sur la 3 couches à la fois. Il en résulte donc une image ne possédant que des attributs sur une seul couleur (la maximal), empêchant alors toutes différenciation optimal entre deux couleur de pixel contigue.

Exemple :

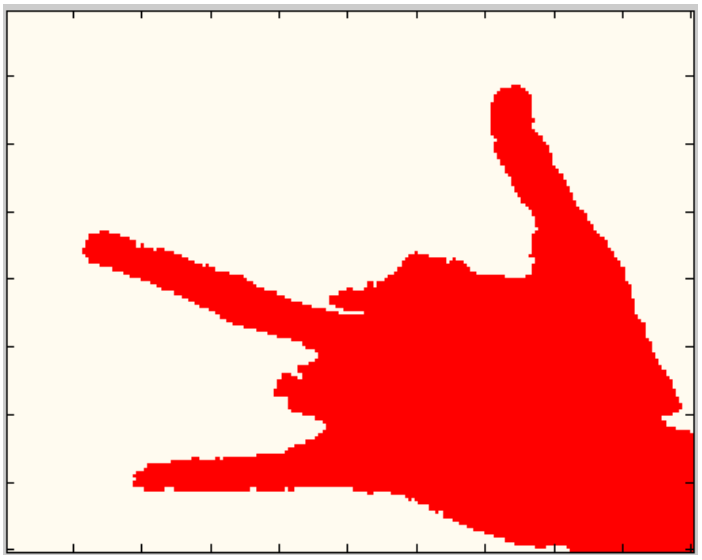
- Soit le Pixel A de couleur [8, 5 235] qui est de couleur BLEU très prononcé.
- Soit le pixel B de couleur [215, 200, 235] qui est de couleur LILAS clair.

Ces deux pixel ont la même valeur max sur la couche BLEU, entraînant après réduction par max à la total identité entre ces deux pixels, qui à l'origine sont bien différents.

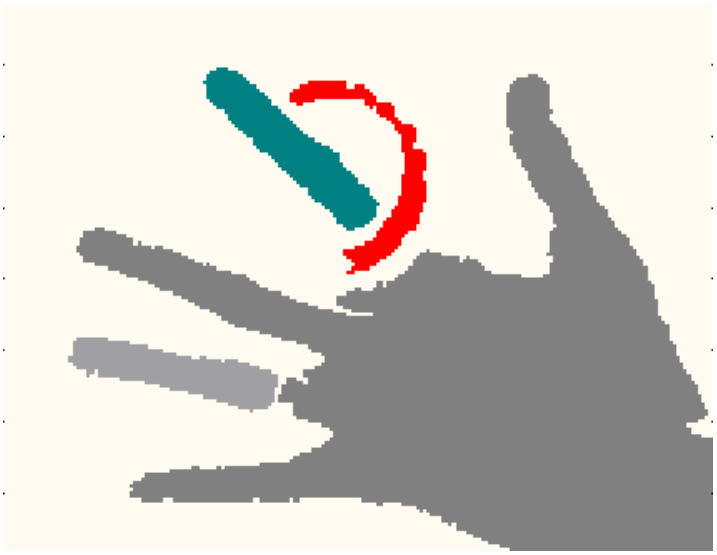
Si nous établissons l'image avec la méthode suivante :

```
I  = (I(:, :, 1) < 128) + ( 4 * (I(:, :, 2) < 128) + 2 * (I(:, :, 3) < 128));
```

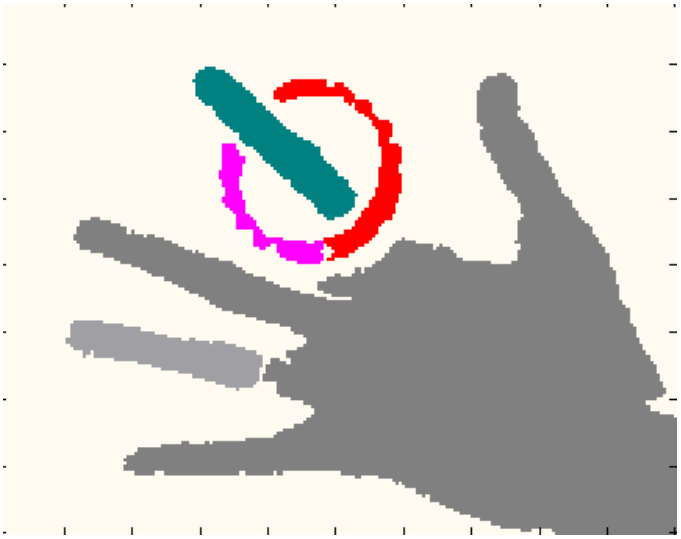
Cette méthode permet d'unifier notre travail sur les différentes couches, en renvoyant dans I une valeur la somme de la valeur d'un pixel rouge, plus 4 fois la valeur d'un pixel vert et 2 fois la valeur d'un pixel bleu, si ces derniers sont inférieurs à 128. Le résultat retourné est compris entre 7 et 889 (le cas ou les 3 couleurs sont à 127 par exemple). Cette opération nous permet d'exprimer un triplet colorimétrique sur une seule valeur, nous auront alors des comparaisons bien plus précises entre deux pixel.



Notre algorithme avec comme valeur de région petite 5%

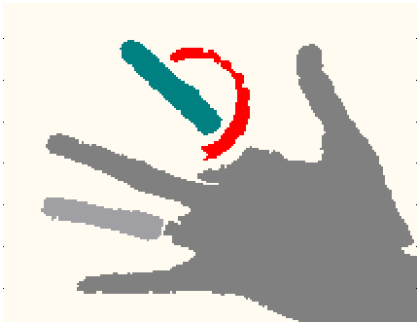


Une fois le seuil baisé à 1%, certaine partie de l'image apparaissent, tel une partie de l'anneau, ou encore l'annulaire.

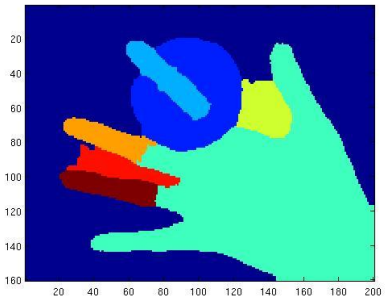


Cette fois ci, avec un seuil de 1 pour 1000, la seconde partie de l'anneau apparaît.

Original



Notre Algorithme

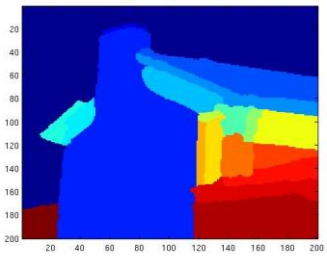


JSeg

JSeg trouve plus de région que dans notre algorithme, surtout les zones entre les doigts. Il amène aussi une segmentation supplémentaire pour l'annulaire que notre algorithme.

Original

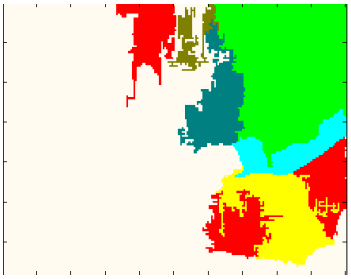
Notre Algorithme



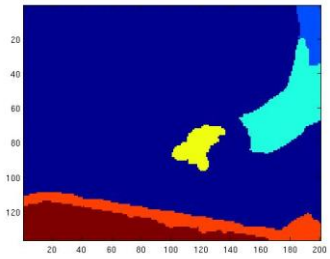
JSeg

Notre algorithme et JSeg ont un ici un rendu quasiment identique, Jseg trouve juste une nouvelle région, et possède un dégradé du toit un peu plus segmenté, mais l'image est sensiblement la même dans son découpage.

Original

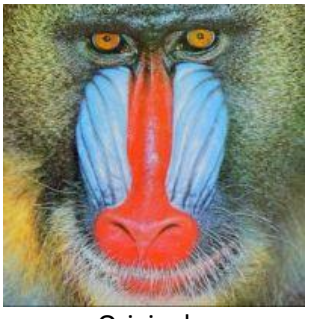


Notre Algorithme

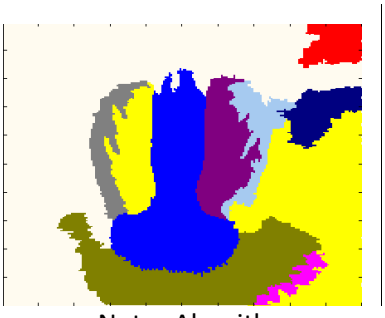


JSeg

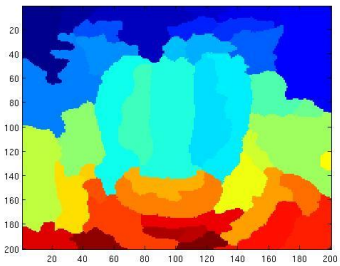
Là ou notre algorithme ne s'en sort quasiment pas, avec une suppression total de la table, main qui sont uniformisé avec le mur, JSeg lui réussi à extraire les éléments les plus importants de l'image que sont la main et la table. La texture du fond de l'image n'aide pas notre algorithme.



Original



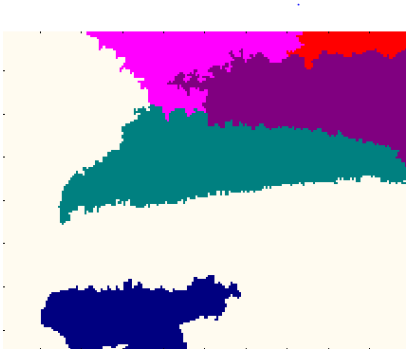
Notre Algorithme

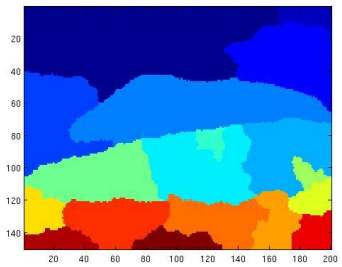


JSeg

JSeg produit ici un résultat plus probant que notre algorithme qui néglige une bonne partie de l'image. JSeg réussi à délimiter les différence de couleur sur le pelage, là ou notre algorithme ne permet de délimiter que sur la forme de la gueule de l'animal.


Original

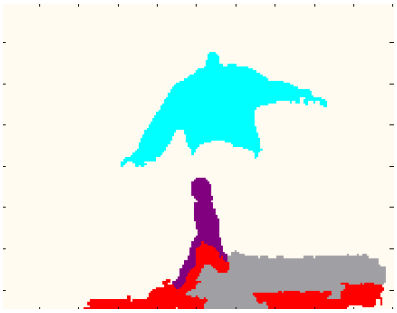

Notre Algorithme

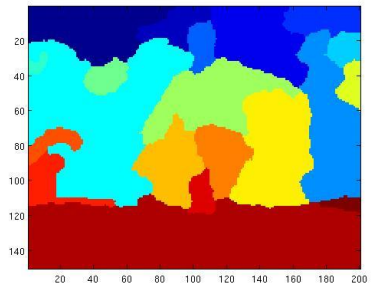


JSeg

Comme dans l'image précédente, le travail de JSeg est un peu plus poussé que celui de notre algorithme, là ou notre algorithme ne permet de lisser les région qu'en fonction de leur couleur, JSeg arrive à les délimiter en fonction de leur texture, amenant un travail beaucoup plus poussé.


Original


Notre Algorithme



JSeg effectue ici un travail plus poussé, mais qui au final nuit au résultat, étant donné que la silhouette du footballeur est noyé dans des zones d'arrière plan qui devrait être négligeable. Cependant, l'on discerne tous de même l'esquisse global du sportif, quand dans notre algorithme, la tête ou encore le maillot a été absorber par le fond.