Université de Bourgogne
Faculté IEM

Travail fait pour le Module:
Imagerie Couleur et Multi-spectrale

Spécialité : Computer vision.

Travail fait par :

Mohammed El Amine MOKHTARI mohammed-el-amine_mokhtari@etu.u-bourgogne.fr

Amine MANSOURI. Amine_Mansouri@etu.u-bourgogne.fr

Année Universitaire : 2020/2021.

Table des matières

Index des figures

Introduction

Matlab est devenu un outil très utilisé dans l'industrie et parmi ces utilisations il y a le traitement d'image.
Matlab contient plusieurs fonctions qui nous facilitent tout travail dans le domaine du traitement d'image, parce qu'on peut considérer une image comme une matrice et puis le nombre des bandes seront traiter comme des dimensions d'une matrice.
Donc l'objectif du ce travail est de se familiariser avec Matlab dans le domaine du traitement d'image donc on va prendre en œuvre des méthodes de segmentation et de classification des pixels d'une image.

Partie I : Initiation à Matlab pour l'image

1- Découverte de l'environnement de travail :

Le rôle de chaque fonction :
dir :
ls :
cd : pour changer le répertoire
clear all : pour effacer toute les variables qui sont l'espace de travail
close all : pour fermer toute les fenêtres ouverte
clc : pour nettoyer la fenêtre de commande

2- Déclaration et initiation de variables :

a- Pour déclarer un scalaire on fait comme ça : **y = 1**
b- Pour déclare un vecteur ligne on fait : **y = [1 3 2]** et pour un vecteur colonne on fait : **y = [1; 3; 2]**
c- Pour déclare une matrice on fait : **y = [1 3 6 ; 8 4 7] ;**
Le rôle du point-virgule e fin de linge c'est pour ne pas afficher la valeur de cette matrice dans la fenêtre de commande.
d- y(2 , 2) = **4**
e- Le rôle de whos :
f- Pour créer une matrice 10 lignes, 10 colonnes et 3 bandes on va utiliser la fonction zeros donc on va la mettre de la façon suivante :
Y = zeros(10, 10, 3)
Le premier paramètre va contrôler les lignes, le deuxième va contrôler les colonnes et le dernier pour contrôler les bandes

Donc le programme pour faire varier les valeurs des pixels sur les différentes bandes sera le suivant :

```
clear all; close all; clc
Y = zeros(10,10,3);

Y(5,5,1) = 10
Y(5,5,2) = 50
Y(5,5,3) = 150
```

Et après exécution on obtient les bandes suivantes :

```
Y(:,:,1) =

     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0    10     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
```

Bande 1

```
Y(:,:,2) =

     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0    50     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
```

Bande 2

```
Y(:,:,3) =

     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0   150     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
```

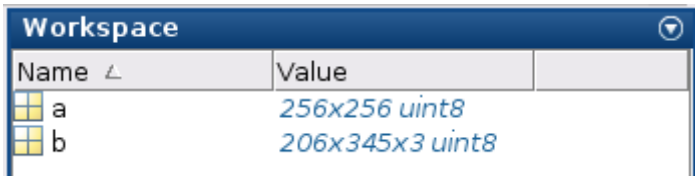
Bande 3

4- Manipulation des images :

2) Les dimensions et le type de a :

Comme on peut voir sur la photo précédente, les dimensions de la photo sont : 256 pixels par 256 et pour le type : c'est entier non-signé en 8 bits.
Ce qui veut dire que c'est une image au niveau de gris (noire et blanc)

4) Les dimensions et le type de b :



Pour la photo **b** on remarque que les dimensions sont 206 pixels par 345 pixels par 3 bandes ce qui veut dire que c'est une image couleur (RGB).

5) Affichage des dimensions en utilisant la fonction size :

```
>> size(b)

ans =

   206   345     3
```

Donc on remarque qu'on a obtenu les mêmes résultats.

6) Application d'algorithme à l'image :

La commande qu'on doit appliquer est la suivante :

```
a2 = log(single(a));
```

Donc on va vous montrer l'image **a** avant et après l'algorithme :

Figure 1: Image avant l'algorithme

Figure 2: Image après l'algorithme

7) Lorsqu'on a voulu afficher l'image après l'application du l'algorithme on va avoir une erreur au niveau d 'affichage parce que après qu'on fait des calcules mathématique sur l'image on obtient une matrice donc pour l'afficher il faut la transformer à une image c'est pour ça il faut ajouter les symboles **'[ ]'** dans les paramètres de la fonction **imshow( )** .

Donc voila le code qu'on a fait pour obtenir la photo après algorithme :

```
a2 = log(single(a));

figure, imshow(a)

figure, imshow(a2,[])
```

8) Donc après qu'on a appliqué la fonction algorithme sur notre image, on va essayer de revenir à notre image en appliquant la fonction exponentielle.
Donc voila le code qu'on a appliqué :

```
a3 = exp(a2);
```

9) Après application de la fonction exponentielle sur la nouvelle image on va l'afficher en utilisant la même méthode qu'on a fait pour afficher l'image après la fonction algorithme.

Figure 3: L'image a après la fonction exponentielle

10) L'erreur c'était toujours au niveau d'affichage parce que l'image est devenu une matrice après le calcule mathématique donc il faut ajouter les symboles **'[ ]'** dans les paramètres de la fonction **imshow( ).**

11) Le rôle de chaque fonction :

Impixelinfo : Elle nous donne le niveau de gris du pixel choisis par le curseur.

```
h = imshow(a);
hp = impixelinfo;
```

Après qu'on a appliqué cette fonction on obtient le résultat suivant :

La fonction impixelinfo( )

Comme vous voyez on trois valeurs : la première c'est pour la ligne du pixel sélectionné, la deuxième c'est pour la colonne du pixel sélectionné et la dernière c'est pour la valeur de luminance de ce pixel.

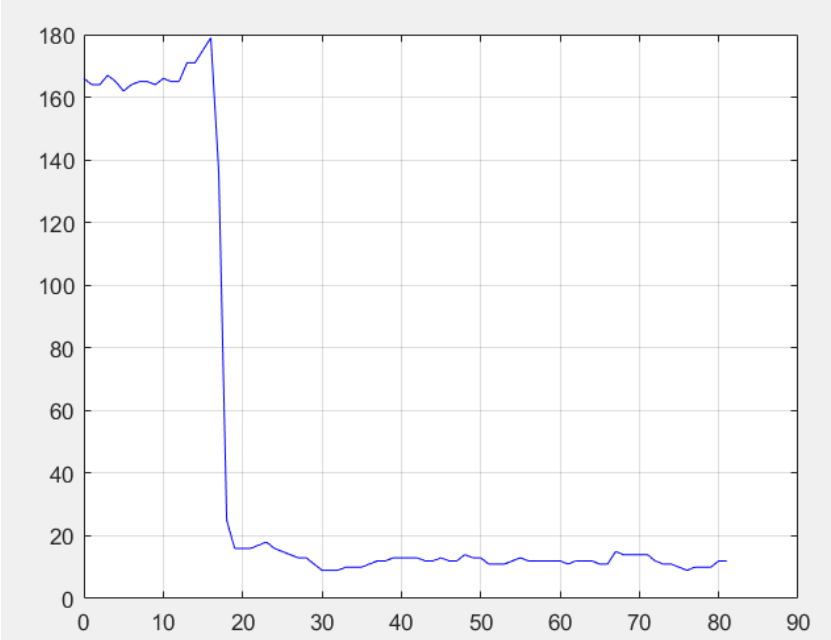
improfile: c'est tracer une ligne et prendre le niveau de gris de chaque pixel qui se trouve sur cette ligne et les tracer sur un graphe.
x -> la position x du pixel
y -> la valeur d niveau du gris de ce pixel

Donc la première des choses il faut tracer la droite en utilisant le code suivant :

```
I = imread('images/cameraman.tif');
x = [19  100];
y = [100 100];
```

Donc comme a dit cette fonction va tracer une droite sur l'image à partir des valeurs X et Y alors on obtient notre image plus la droite :

Après qu'on a tracer la droite rouge, la fonction improfile va prendre les valeurs de la luminance et les tracer dans le graphe suivant :



Comme on peut remarquer que pour les première valeur on a un luminance importante par rapport à les autres et tout ça il est du au droite qu'on a tracer parce que la droite a commencé dans une zone plus blanche puis une zone plus grise c'est pour ça on voit que les valeurs du luminance ont changé directement d'une valeur de 160 à une valeur de 10.

imdistline : `Elle nous permet de calculer la distance entre deux points dans une image par unité de pixel.`

Donc pour utiliser cette fonction on a fait le code suivant :

```
imshow('images/cameraman.tif');
h = imdistline(gca,[10 100],[10 100]);
```

Alors cette fonction va nous génère une droite pour calculer la distance entre deux pixels :

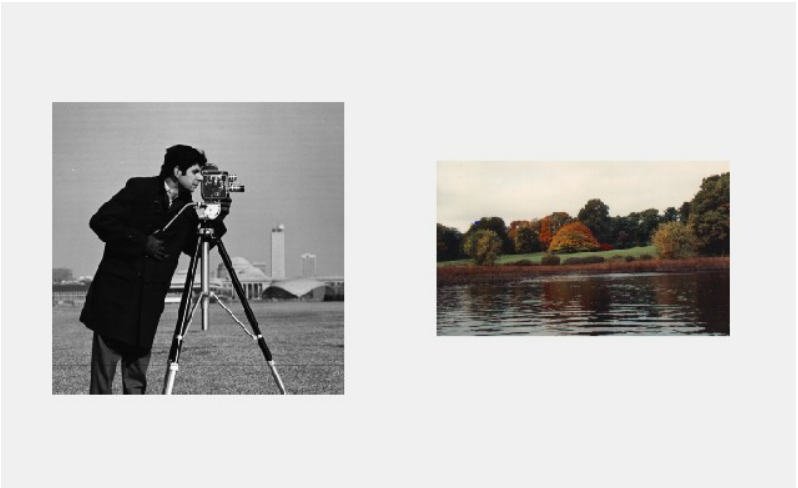
Comme vous voyez, on a pu calculer la distance entre les deux points en utilisant la fonction **imdistline**.

Le rôle de la fonction subplot :

```
figure(10)
subplot(1, 2, 1), imshow(a)
subplot(1, 2, 2), imshow(b)
```

En faite cette fonction va deviser le plans de la figure en une ligne et deux colonnes, puis le troisième paramètre va contrôler dans quel demi-plan on veut mettre telle photo.

Donc pour notre cas on obtient la figure suivante :



Vous pouvez remarquer que le plan est devisé en deux, et on mis la première photo dans le premier demi et la deuxième photo dans le deuxième demi.

13) Pour obtenir un carré blanc au milieu de l'image, il faut juste changer les valeurs de ces pixels et les mettre en 255 qui est la valeur maximal qu'un pixel peut prendre.

```
taille = size(b)
Tx = floor(taille(2)/2);
Ty = floor(taille(1)/2);

b(Ty-15:Ty+15, Tx-25:Tx+25 , :) = 255
```

On doit calculer le milieu de cette image pour les pixels des lignes et les pixels des colonnes, donc **Tx** c'est le milieu des lignes et **Ty** c'est le milieu des colonnes.
Puis il suffit juste d'ajouter des petites valeurs à ces milieux pour qu'on obtienne un carré de 50*30.

Alors on a appliqué cet algorithme sur les 3 bandes couleurs comme vous voyez.

Et voilà l'image avec le carré blanc :

14) Après toutes les opérations on peut enregistrer la nouvelle image en utilisant la fonction **imwrite ( ).**

```
imwrite(b, 'autumn_modif.tif')
```

Et comme ça l'image sera enregistrée sur le même répertoire de notre projet.

Partie II : Traitement des images couleur : Intensité, luminance et composantes couleur :

1) Transformation d'une image de l'espace RGB à une image au niveau de gris :

Donc pour faire ça, il faut calculer la moyenne des trois bandes pour chaque pixel, alors on a fait le programme suivant :

```
a = imread('images/Fleur.tif');
[w, h, l] = size(a)
b = zeros(w, h);

for n = 1:w
    for m = 1:h
        b(n, m) = (a(n, m, 1) + a(n, m, 2) + a(n, m, 3)) / 3;
    end
end
```

Tout d'abord on a déclaré une matrice monodimensionnelle avec les dimensions de la matrice b pour qu'on stocke les nouvelles valeurs dedans.

Puis on la petite boucle qui va traverser toute la matrice pour qu'on fasse le calcul des moyennes de tous les pixels.

Après qu'on a fait ce calcul on a obtenu l'image suivante :

Figure 4: L'image originale

Figure 5: L'image après calcul

-On remarque que la photo n'est pas vraiment lisible, donc pour régler ce problème on doit multiplier la valeur de luminance de chaque pixel par un coefficient.

Donc voilà le résultat qu'on obtient :

Figure 6: L'image après multiplication

-On remarque que l'image est plus claire maintenant après qu'on l'a multiplié par les coefficients.

-Donc maintenant on va utiliser la fonction **rgb2gray ( )** qui est déjà intégrée dans Matlab :

```
b2 = rgb2gray(a)
figure, imshow(b2)
```

Après exécution on trouve la photo suivante :

On peut bien remarquer que l'image est déjà très nette ce qui veut dire que la fonction **rgb2gray** contient les deux opérations qu'on a fait (le calcul des moyennes de trois bande, multiplication par les coefficients).

2) Quel espace pour le traitement d'image ?

Si on veut faire un traitement d'image sur un espace ou les composantes sont non corrélés donc il faut choisir impérativement l'espace HSV, parce que c'est le seul espace ou on n'a pas une relation entre les composantes ce qui veut dire que chaque composantes est indépendantes de l'autre donc si on fait un traitement sur une des composantes ça n'influent pas sur les autres composantes.

3) Faire des traitements sur l'espace HSV :

-Si on veut rendre une image plus claire dans l'espace HSV il faut juste multiplier la bande H (première bande) par un coefficient choisi pour nous.

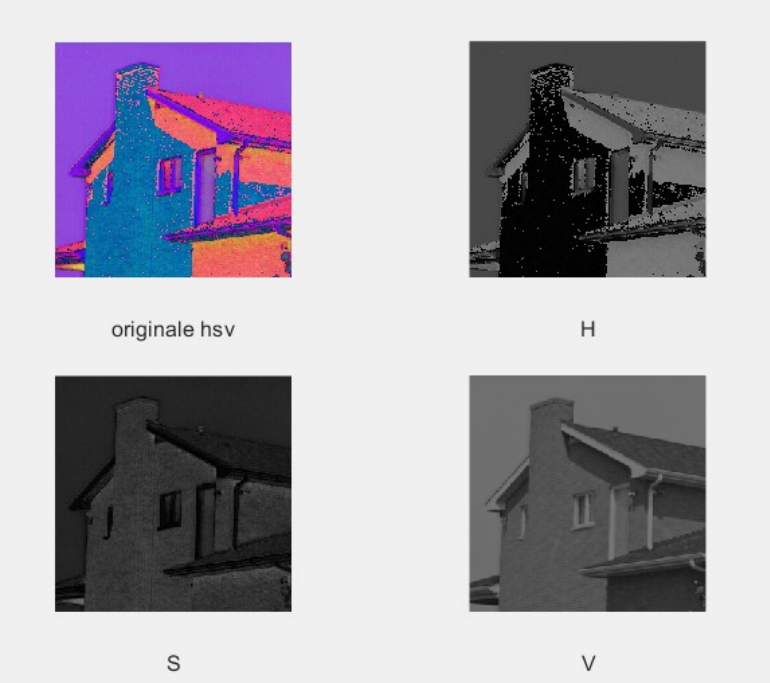
-Si on veut rendre les couleurs plus vives dans l'espace HSV il faut juste multiplier la bande S (la deuxième bande) par un coefficient choisi par nous.

-Si on veut changer la teinte de l'image dans l'espace HSV il faut multiplier la bande V (la troisième bande) par un coefficient.

Donc on a fait le programme suivant en prenant le coefficient = 0.5 :

```
house = imread('images/house.tif');
houseHSV = rgb2hsv(house);
f = 0.5
newHSV1 = houseHSV(:,:,1)*f;
newHSV2 = houseHSV(:,:,2)*f;
newHSV3 = houseHSV(:,:,3)*f;
subplot(2,2,1)
imshow(houseHSV)
xlabel('originale hsv')
subplot(2,2,2)
imshow(newHSV1)
xlabel('H')
subplot(2,2,3)
imshow(newHSV2)
xlabel('S')
subplot(2,2,4)
imshow(newHSV3)
xlabel('V')
```

Et comme résultat on obtient les images suivantes :



-En utilisant matlab, on peut faire pas mal des conversions des images RGB donc pour voir les différents types de conversion on peut utiliser la fonction ***makecform*** .

Les autres espaces couleurs :

CMYK : pour Cyan Magenta Yellow(jaune) et Key(noir) : cette espace couleur est utilisé par les imprimeurs commerciaux donc les imprimantes sont basées sur cet espace couleur.

XYZ : C’est un espace défini par la commission internationale de couleur pour constituer une amélioration sur l’espace RGB.

LAB : est un espace de couleur particulièrement utilisé pour la caractérisation des couleurs de surface.

Maintenant on va faire la conversion de notre image dans ces espaces là en utilisant la fonction **makecform.**

UVL : c’est l’espace ultraviolet

1-On va faire la conversion de notre image de l’espace RGB à l’espace LAB :

```
cform = makecform('srgb2lab');
lab = applycform(a,cform);
figure
imshow(lab, [])
```

Figure 7: La fonction makecform pour l’espace LAB

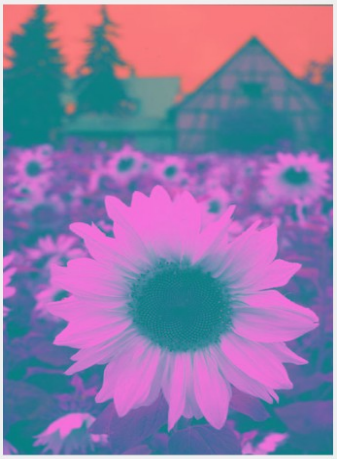


Figure 8: La même image dans l’espace LAB

2-On va faire la conversion de notre image de l’espace RGB à l’espace XYZ :

```
cform = makecform('srgb2xyz');
xyz = applycform(a,cform);
figure
imshow(xyz, [])
```

Figure 9: Le code qu'il faut appliquer

Figure 10: La même photo dans l'espace XYZ,

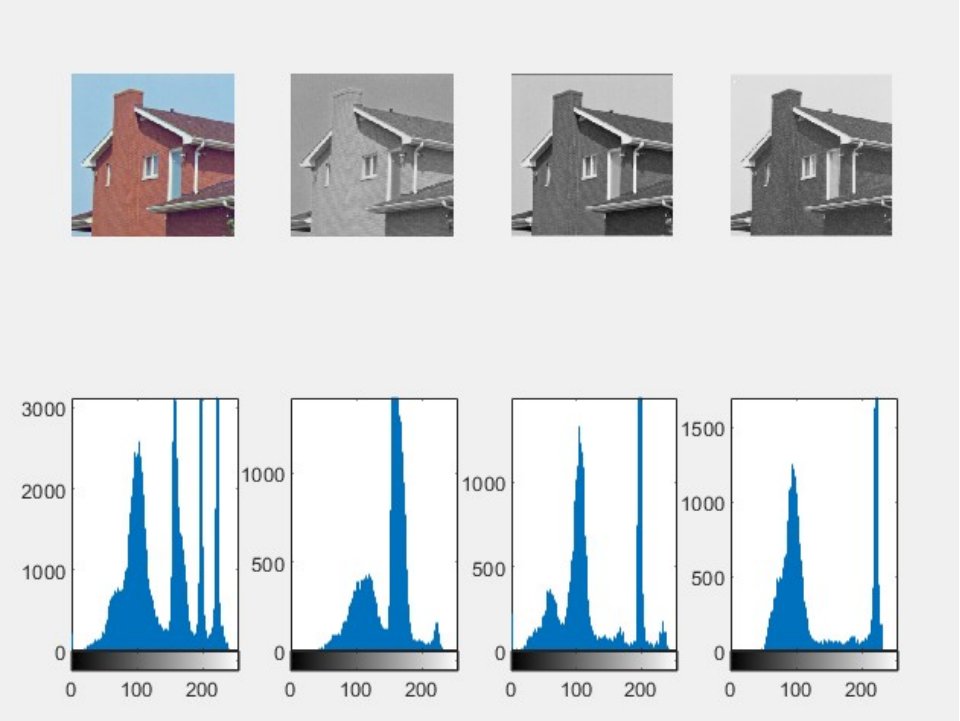
Partie III : Seuillages sur les composantes de couleur

1/ On désire isoler la façade de la maison :

2) Affichage de l'image couleur plus les trois composantes

```
im = imread('images/house.tif');
figure(1)
subplot(1,4,1); imshow(im)
subplot(1,4,2); imshow(im(:, :, 1))
subplot(1,4,3); imshow(im(:, :, 2))
subplot(1,4,4); imshow(im(:, :, 3))
```

3) L'histogramme des trois composantes R, G, B



4) Filtrage de l'image pour avoir que la façade :

Si on remarque bien dans les histogrammes, on remarque que pour chaque bande on a un pique qui signifie la valeur de tell couleur sur l'image, donc si on veut laisser que la façade de la maison alors il faut laisser que les piques correspondent à la couleur de la façade qui est le rouge.

Donc on peut créer des intervalles disant que : on prend les valeurs entre 150 et 190 pour la bande rouge comme ça on va prendre que le pique qui donne la couleur rouge. Et pour la bande verte on la laisse comme elle est et finalement pour la couleur bleue il faut l'enlever parce que la façade n'a pas de

couleur bleue plutôt la couleur bleue est donne le font c'est pour cela il faut l'enlever et pour faire ça on va éliminer les valeurs supérieures à 200 car le pique du bleu commence à 200.

Donc on va transformer tous ce qu'on a dit à un programme matlab, mais comme on a dit qu'on veut avoir la façade de la maison donc il faut convoluer les masque qu'on va obtenir avec l'image originale pour avoir que la façade.

```
masque =im(:,:,1)> 150  & im(:,:,1)<190 & im(:, :,3) < 200 ;
figure, imshow(im)
facade_rouge = uint8(masque).*im;
imshow(facade_rouge)
```

Donc après qu'on applique ce masque sur l'image originale on trouvera la façade comme suite :

La façade de la maison

On peut bien remarquer qu'on a obtenu que la partie façade de la maison qui est la partie rouge, et pour les pixels noirs ils sont du au masque qu'on a fait parce que le masque va garder que les pixels de la façade parce qu'on a fait une **ET logique** et qui va sortir soit **0** soit **1** alors pour les pixels de la façade on va les multiplier par 1 pour garder leurs mêmes valeurs par contre les autres pixels on va les multiplier par **0** pour les enlever.

5) Donc maintenant pour avoir la façade bleue, on faire les étapes suivantes :

5-a- On va changer la couleur de la façade à partir de l'image qu'on a obtenu (l'image de la façade seule), veut dire qu'on va éliminer toutes les composantes couleurs sauf la composante bleue pour avoir juste la façade on bleue.

- Le programme :

```
facade_bleue = facade_rouge;
facade_bleue(:, :, 1)= 0;
facade_bleue(:, :, 2)= 0;
facade_bleue(:, :, 3) = facade_bleue(:, :, 3);

figure
imshow(facade_bleue)
```

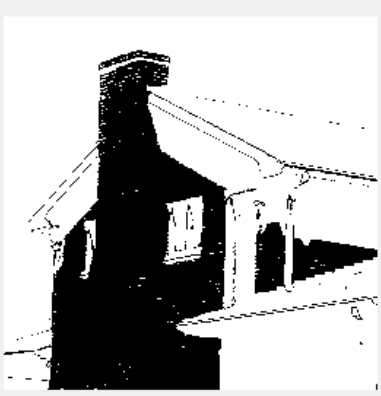
- La façade bleue :

5-b- Puis on va inverser le max qu'on a obtenu (voir la façade)

- Le programme :

```
masque_inverse = 1-masque;
imshow(masque_inverse)
```

- L'image :



5-c- Puis après on va convoluer le masque inversé avec l'image originale pour mettre tous les pixels de la façade à zéro.

- Le programme :

```
image_sans_facade = uint8(masque_inverse).*im;
figure
imshow(image_sans_facade)
```

- L'image :

5-d- Et finalement on va sommer l'image obtenu sans façade avec la façade bleue qu'on a obtenu dans la première étape.

- Le programme :

```
image_avec_facade_bleue = image_sans_facade + facade_bleue;
imshow(image_avec_facade_bleue)
```

- L'image finale avec la façade bleue :

6- Transformation de l'image RGB vers HSV :

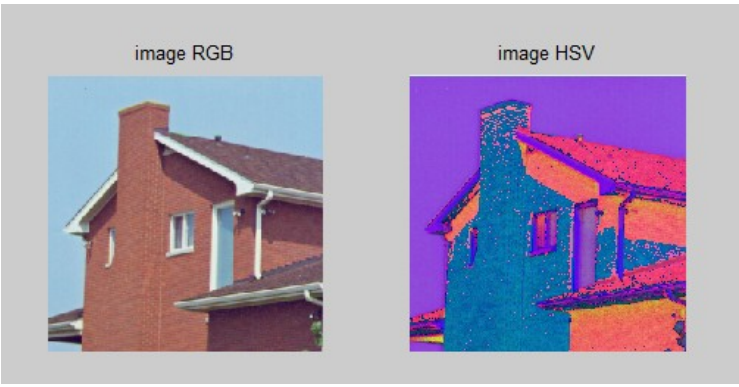


Figure 11: image RGB et HSV.

7) affichage de l'image HSV et les 3 composantes :

avec le programme suivant :

```
%% 7) affichage de l'image couleur + les trois images composantes de HSV:
figure,
subplot(1, 4, 1); imshow(im_hsv);   title('image originale HSV');
subplot(1, 4, 2); imshow(im_hsv(:, :, 1)); title('composante H')
subplot(1, 4, 3); imshow(im_hsv(:, :, 2)); title('composante S')
subplot(1, 4, 4); imshow(im_hsv(:, :, 3)); title('composante V')
```

on obtiens :

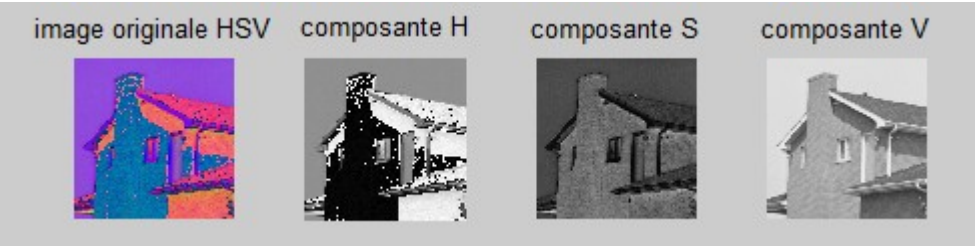


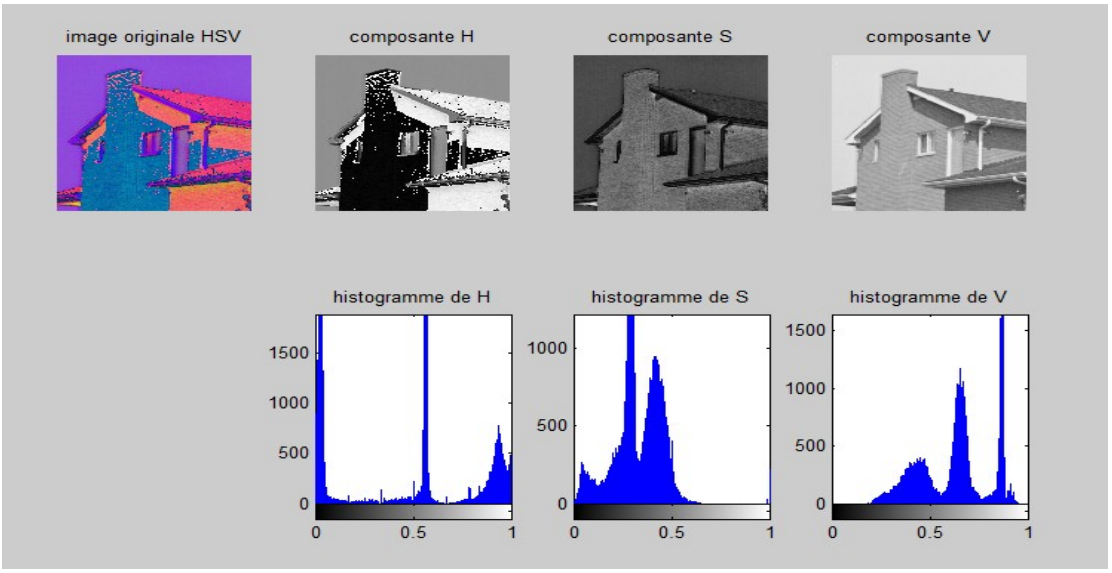
Figure 12: les 3 composantes de l'espasce HSV.

8) affichage des histogrammes correspondant aux composantes H, S et V :

on utilise le code suivant :

```
%% 7) affichage de l'image couleur + les trois images composantes de HSV:
figure,
subplot(2, 4, 1); imshow(im_hsv);   title('image originale HSV');
subplot(2, 4, 2); imshow(im_hsv(:, :, 1)); title('composante H')
subplot(2, 4, 3); imshow(im_hsv(:, :, 2)); title('composante S')
subplot(2, 4, 4); imshow(im_hsv(:, :, 3)); title('composante V')
%% 8) histogrammes:
subplot(2, 4, 6); imhist(im_hsv(:, : , 1)); title('histogramme de H')
subplot(2, 4, 7); imhist(im_hsv(:, : , 2)); title('histogramme de S')
subplot(2, 4, 8); imhist(im_hsv(:, : , 3)); title('histogramme de V')
```

on obtiens l'affichage suivant :



9) le problème que pose l'affichage en couleur de l'image HSV :
les couleurs se rapprochent pour l'utilisateur, on arrive pas a distingué l'hombre si on voit cette image hsv pour la première fois.

10) la bande qui élimine les hombres est la bande **H**, sur les autres bandes on peut remarqué la présence de l'hombre.

11) utilisation de la bande H et son histogramme pour faire un seuillage qui met en évidence la façade de la maison :

avec le code suivant :

```
%% 11) utiliser la bande H pour un seuillage:
%%% mask 1
im_mask1 = im_hsv(:, :, 1) < 0.075;
figure, imshow(double(im_mask1)); title('seuillage sur bande H');
```

on obtiens :

Figure 13: seuillage sur la bande H

12) pour éliminer les points singuliers résiduels aprés le seuillage, on utilise les filtres morphologiques suivants : **érosion et dilataion.**

On utilise le code suivant :

```
%%  erosion:
se = strel('square',2);
eroded_mask = imerode(im_mask1_1, se);
%% dilation:
se = strel('square',3);
dilated_mask = imdilate(eroded_mask, se);
figure, imshowpair(eroded_mask, dilated_mask, 'montage'); title('eroded .....  dilated')
```

on obtiens les résultats suivants:

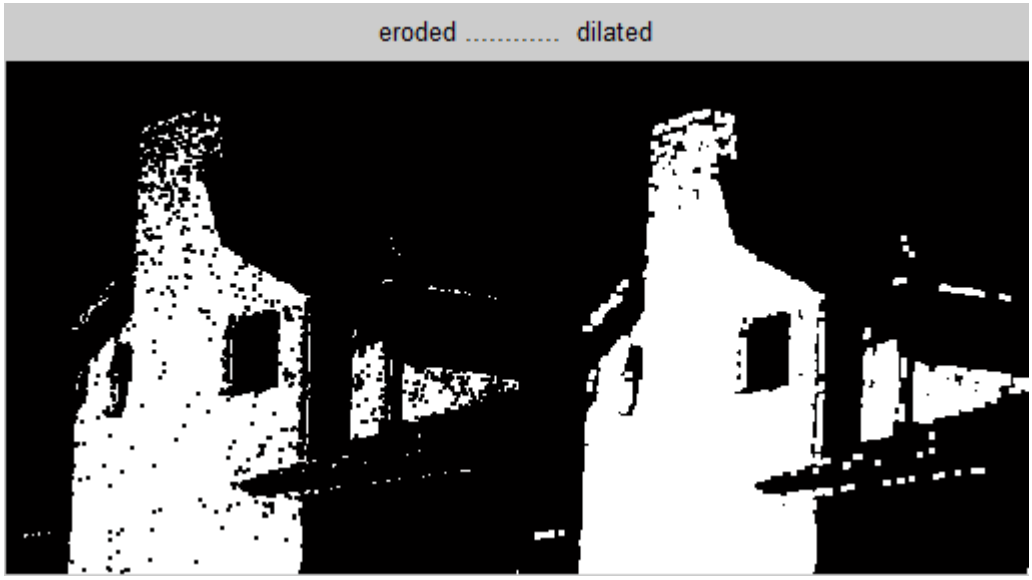


Figure 14: erosion et dilatation du masque de la bande H

13) en utilisant les résultats du seuillage, on recompose une image couleur RGB segmentée qui montre seulement la façade.

On utilise le code suivant :

```
%% ----Applying the mask1_1 on hsv image:
im_hsv(:, :, 1) = im_hsv(:, :, 1).* im_mask1_1;
im_hsv(:, :, 2) = im_hsv(:, :, 2).* im_mask1_1;
im_hsv(:, :, 3) = im_hsv(:, :, 3).* im_mask1_1;
figure, imshowpair(im_hsv, hsv2rgb(im_hsv), 'montage'); title('HSV ..................... RGB');
```

on obtiens le résultat suivant :

14) pour inclure les le bleu sur la façade et les hombres, on travail sur l’espace HSV et on utilise cette fois-ci le masque qui résulte du seuillage de la bande S.

2/ multi-seuillage :

1-on ouvre l'image **hand.jpg** en utilisant le code suivant :

```
%% 1)
im_org = imread('images/hand.jpg'); figure, imshow(im_org);
```

on obtient la figure suivante :

Figure 15: lecture et affichage de l'image hand.jpg

2-on calcule les **coefficients de corrélations** entre chaque couple de bandes (R/G, R/B, G/B):

```
%% 2) coefficients de  correlations:
[r1, p1] = corrcoef(R, G)
[r2, p2] = corrcoef(R, B) % less correlated
[r3, p3] = corrcoef(G, B) % most correlated
```

Figure 16: coeff de corrélation

on obtiens les résultats suivants :

```
r1 =

    1.0000    0.7181
    0.7181    1.0000

r2 =

    1.0000    0.4566
    0.4566    1.0000

r3 =

    1.0000    0.9245
    0.9245    1.0000
```

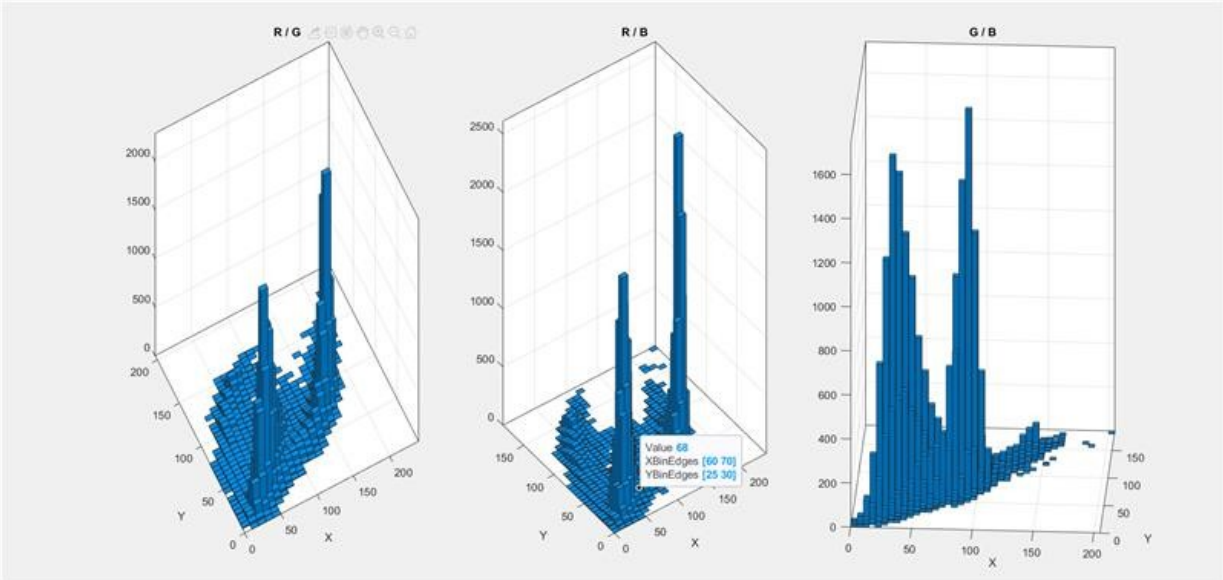
on remarque que le coefficient **r2** correspondant au couple **R/G** constitue le plus décorrélé et le couple **r3** correspondant au couple **G/B** constitue le couple le plus corrélé.

3- les histogrammes bi-dimensionnels des bandes :

```
%% 3) histogram 2D
subplot(1, 3, 1); histogram2(R, G); xlabel('X');    ylabel('Y'); title('R / G');
subplot(1, 3, 2); histogram2(R, B); xlabel('X');    ylabel('Y'); title('R / B');
subplot(1, 3, 3); histogram2(G, B); xlabel('X');    ylabel('Y'); title('G / B');
```

Figure 17: les histogrammes bi-dimensionnelles

on obtiens les résultats



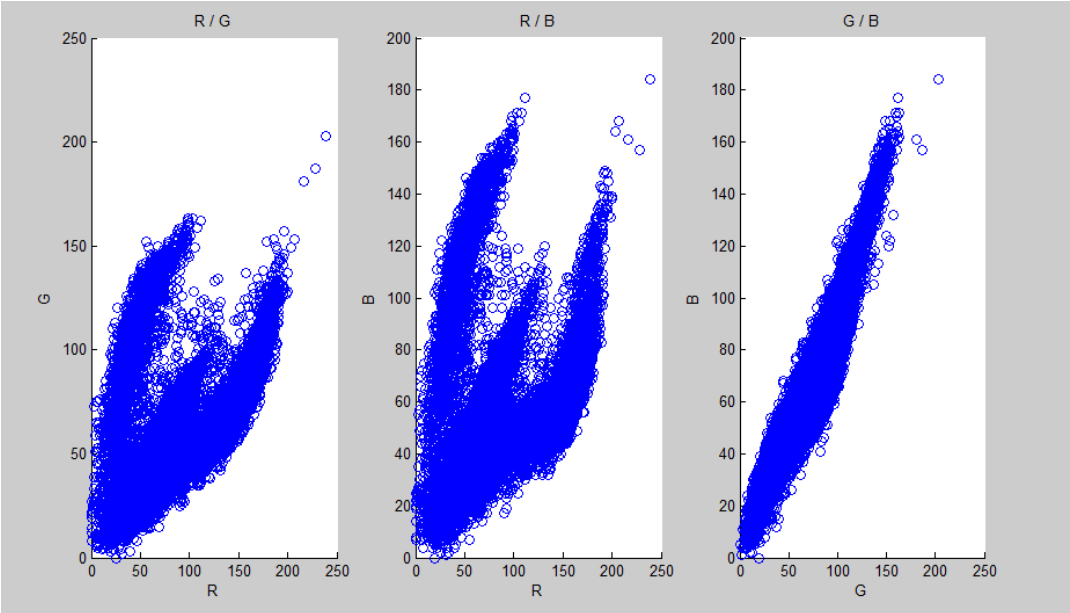
4- le lien entre les coefficients de corrélation et les histogrammes :

si le coefficient n'est pas très importante, alors les piques des histogrammes s'éloignent et et forment deux populations distinctes, se qui va nous aider à mieux discriminer et segmenter l'image.
On remarque dans le cas contraire, en l'occurrence un coefficient de corrélation important, les piques des histogrammes se rapprochent. Par conséquence, il sera difficile de segmenter.

5- Pour afficher chaque couple de bande en utilisant **scatter(),** on utilise le code suivant **:**

```
%% 5)  affichage avec scatter:
%Transformer les matrices en vecteurs:
reshaped_R = reshape(R, [], 1);     reshaped_G = reshape(G, [], 1);     reshaped_B = reshape(B, [], 1);
%Affichage
subplot(1, 3, 1); scatter(reshaped_R, reshaped_G); xlabel('R');    ylabel('G'); title('R / G');
subplot(1, 3, 2); scatter(reshaped_R, reshaped_B); xlabel('R');    ylabel('B'); title('R / B');
subplot(1, 3, 3); scatter(reshaped_G, reshaped_B); xlabel('G');    ylabel('B'); title('G / B');
```

on obtiens en résultat :



6- dans les 2 couples de bandes, on distingue :

-bande R/B : il y a 3 classe (population).

-bande R/G : il y a 2 classe.

7-définir les équations des droites pour le couple R/B :

```
%% 7) Equations des droites:
%%%du graph R/B, on a les points (Xij, Yij):
%%% X[x11 , x12, x13; x21 , x22, x33]  ; Y[y11 , y12, y13; y21 , y22, y23]

%%Extraction manuelle des points du graph:
X = [6, 18, 18;  99, 128, 187 ] ;       Y = [44, 6, 6;  170, 112, 91];
figure, scatter(reshaped_R, reshaped_B); xlabel('X');    ylabel('Y'); title('R / B'); hold on ; grid on;

%%Affichage direct, sont passer par les équations:
% plot( [X(1,1), X(2,1)], [Y(1,1), Y(2,1)] , 'm-');
% plot( [X(1,2), X(2,2)], [Y(1,2), Y(2,2)] , 'g-');
% plot( [X(1,3), X(2,3)], [Y(1,3), Y(2,3)] , 'c-');% hold off;

%%% les equations des droites :
%%droite superieur:
poly_1 = polyfit( X(:, 1), Y(:, 1), 1);  x = linspace(0, 255);  y = polyval(poly_1, x);  plot(x, y, 'm');
%%droite du milieu:
poly_2 = polyfit( X(:, 2), Y(:, 2), 1);  x = linspace(0, 255);  y = polyval(poly_2, x);  plot(x, y, 'g');
%%droite inférieur:
poly_3 = polyfit( X(:, 3), Y(:, 3), 1);  x = linspace(0, 255);  y = polyval(poly_3, x);  plot(x, y, 'c');
hold off;
```

Premièrement, en extrait manuellement les coordonnées des points de départ et de fin des droites, on obtiens ses points du graphique scatter() .

Ensuite, on exploite les fonctions **polyfit()** pour avoir les coefficients du polynôme d'interpolation d'ordre 1. Et **polyval()** pour transformer cette équation en valeurs numériques pour pouvoir les afficher.

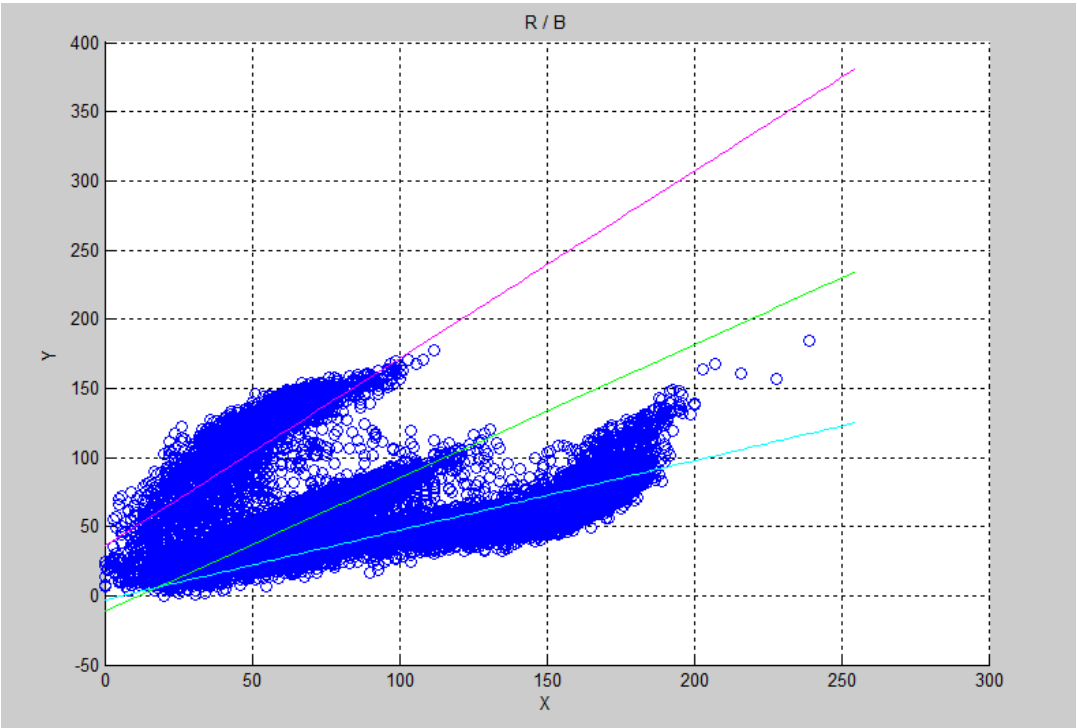


Figure 18: affichage des droites correspondantes aux classes

8- on utilise les équations pour pour segmenter les l'image:

pour faciliter la taches, on choisi la droite du milieu comme une droite qui sépare les 2 classes.

On utilise le code suivant :

```
figure,
cond_11 = B > (poly_1(1) * R + poly_1(2));     subplot(3, 3, 1); imshow((cond_11 )); title('cond-11')
cond_21 = B > (poly_2(1) * R + poly_2(2));     subplot(3, 3, 2); imshow((cond_21 )); title('cond-21')
cond_31 = B > (poly_3(1) * R + poly_3(2));     subplot(3, 3, 3); imshow((cond_31 )); title('cond-31')
%%%
cond_12 = G > (poly_1(1) * R + poly_1(2));     subplot(3, 3, 4); imshow((cond_12 )); title('cond-12')
cond_22 = G > (poly_2(1) * R + poly_2(2));     subplot(3, 3, 5); imshow((cond_22 )); title('cond-22')
cond_32 = G > (poly_3(1) * R + poly_3(2));     subplot(3, 3, 6); imshow((cond_32 )); title('cond-32')
%%%
cond_13 = B > (poly_1(1) * G + poly_1(2));     subplot(3, 3, 7); imshow((cond_13 )); title('cond-13')
cond_23 = B > (poly_2(1) * G + poly_2(2));     subplot(3, 3, 8); imshow((cond_23 )); title('cond-23')
cond_33 = B > (poly_3(1) * G + poly_3(2));     subplot(3, 3, 9); imshow((cond_33 )); title('cond-33')
```

dans ce code, on compare entre une bande de couleur et une autre bande pondérée avec les coefficients des droite, trouvées précédemment.

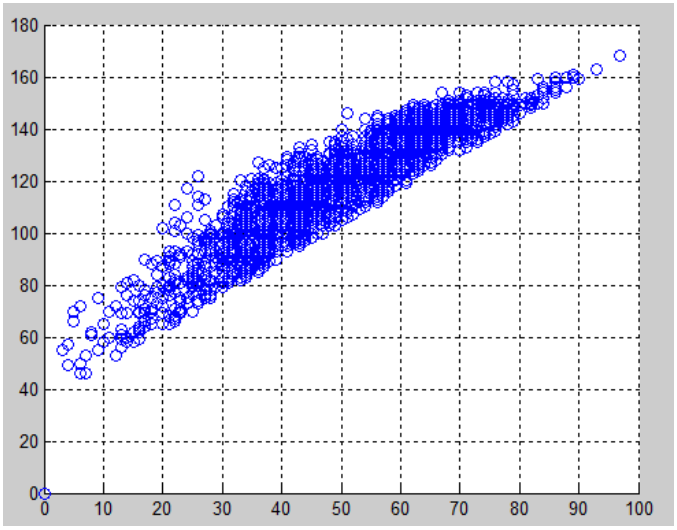
on obtiens le résultat suivant :

On remarque que les masques **cond_11 et cond_31** (respectivement droite supérieur, droite inférieur) du couple de bandes B/R sont les mieux adaptés.

Remarque :
avec ces lignes de code:

```
%%% scatter the upper half of the line:
R2 = reshape((cond_11 .* R), [], 1);     B2 = reshape((cond_11 .* B), [], 1);
figure, scatter(R2, B2);grid on;
```

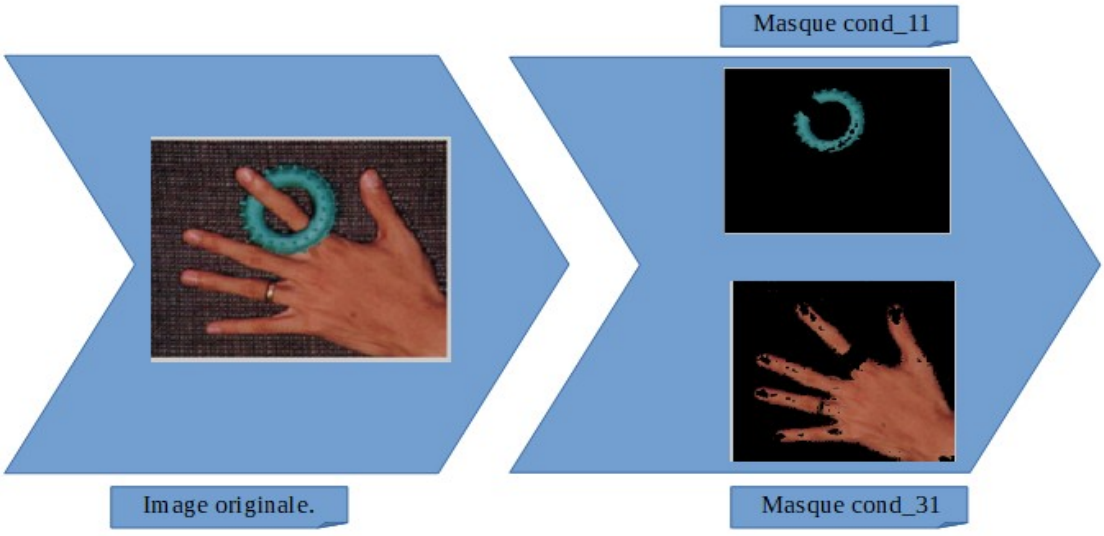
on peut afficher la partie supérieur seulement des pixels, pour comprendre mieux le résultat de l'affichage des masque précédent, comme suite :



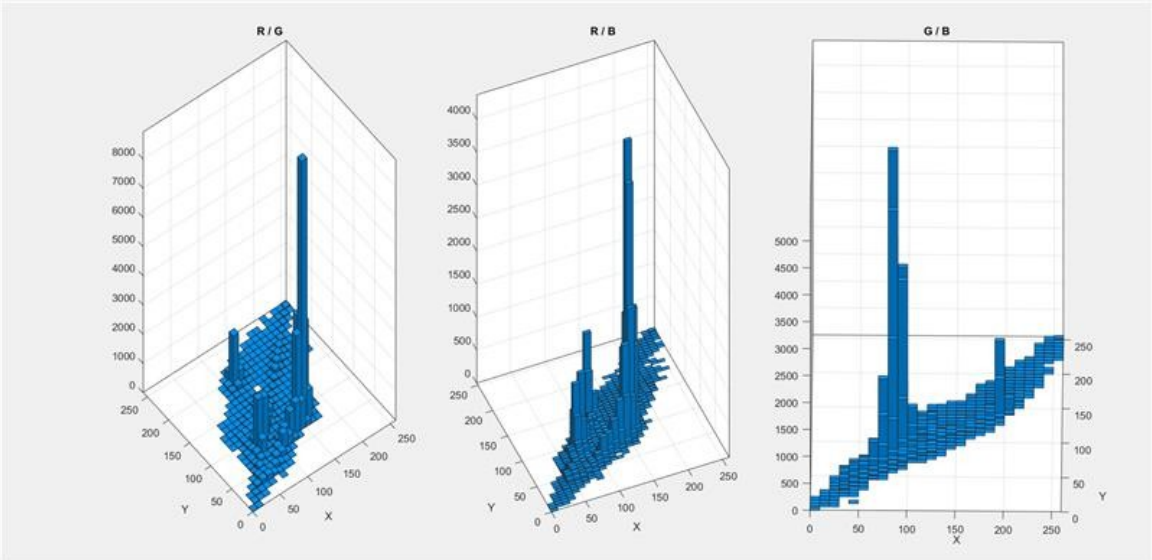
- on peut appliquer ces masques trouvés sur l'image originale, en utilisant ce code suivant :

```
%%%% utiliser la masque cond_11:
im2 = im;
im2(:, :, 1) = double(im(:, :, 1)) .* cond_11;
im2(:, :, 2) = double(im(:, :, 2)) .* cond_11;
im2(:, :, 3) = double(im(:, :, 3)) .* cond_11;
figure, imshow(im2)
%%%% utiliser la masque cond_31:
im3 = im;
im3(:, :, 1) = double(im(:, :, 1)) .* (1-cond_31);
im3(:, :, 2) = double(im(:, :, 2)) .* (1-cond_31);
im3(:, :, 3) = double(im(:, :, 3)) .* (1-cond_31);
figure, imshow(im3)
```

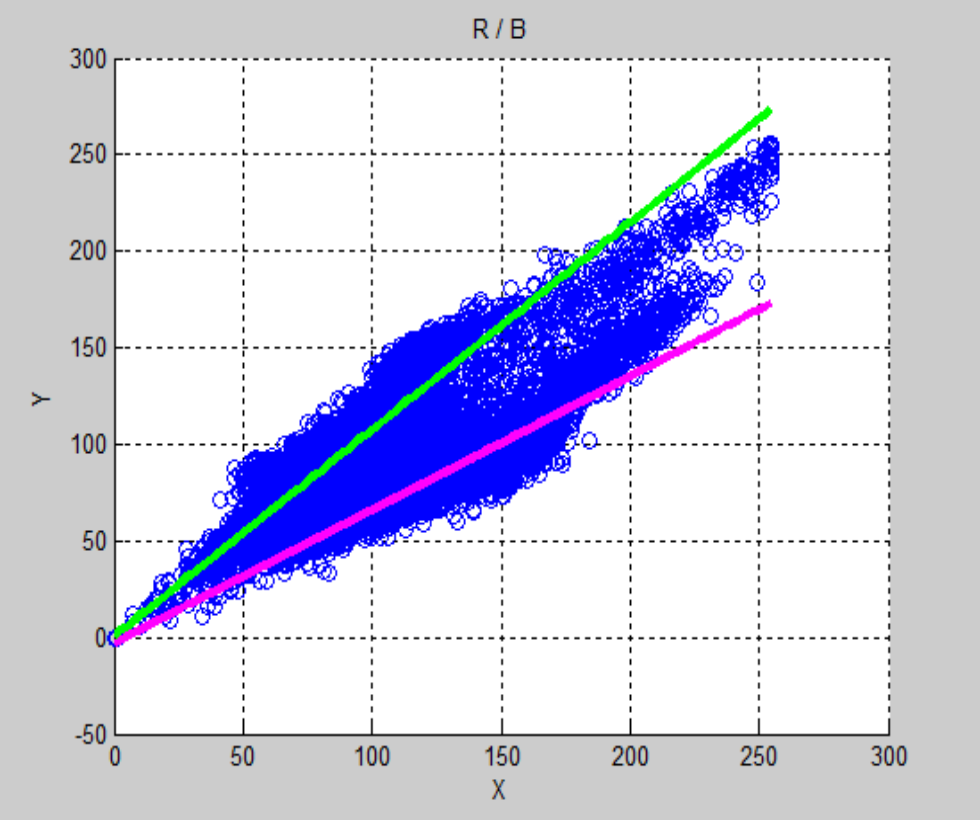
on obtiens le résultat suivant :



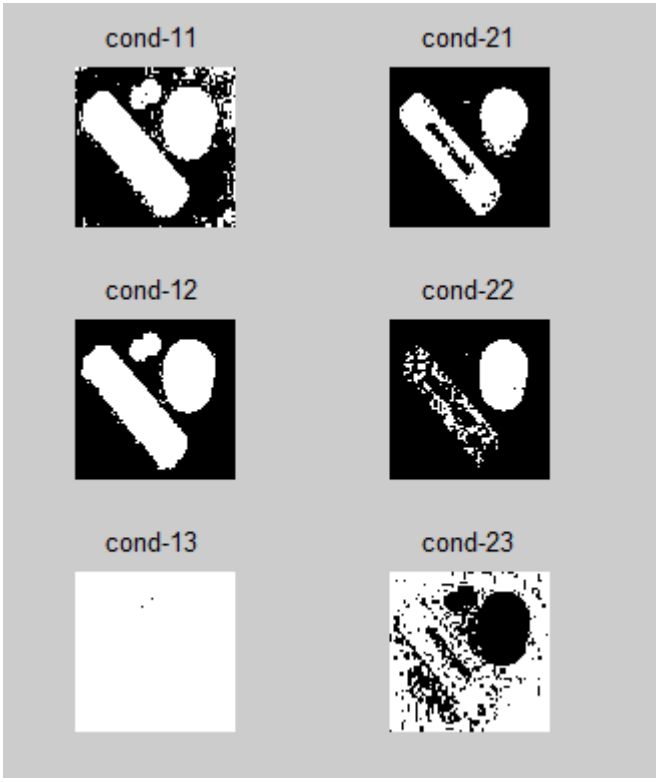
9- On refait le travail avec l'image **15.jpg :**
les histogramme bi-dimensionnelles :



on obtiens pour le couple de bande le plus décorrélé le graph scatter suivant, et on trace les droites :



après l'obtention des équations des droites, on applique la segmentation et on obtiens les masques suivants :



on remarque que le masque **cond_12** donne de meilleur resultats que les autres masques . On applique se masque sur l'image originale et on obtiens :

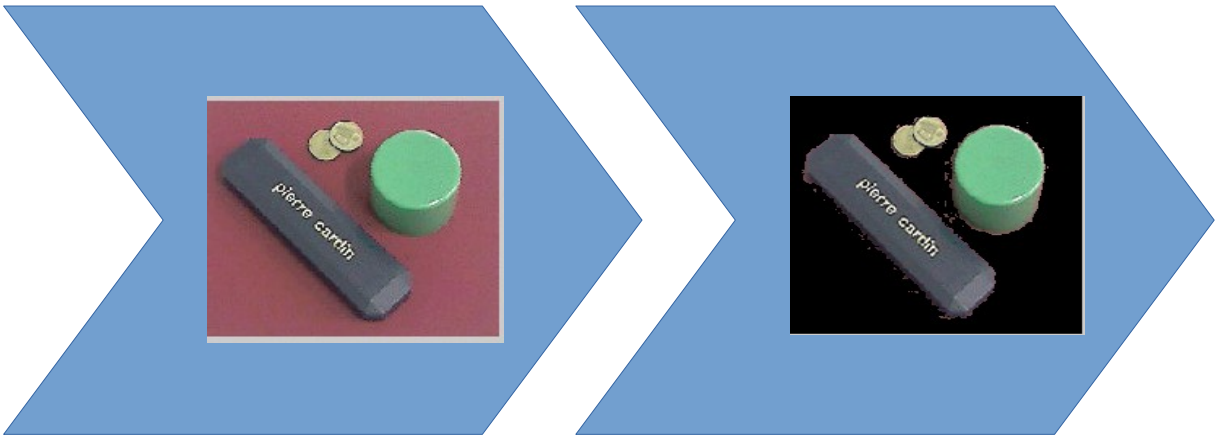


Image originale

Application du masque cond_12

3- Classification :

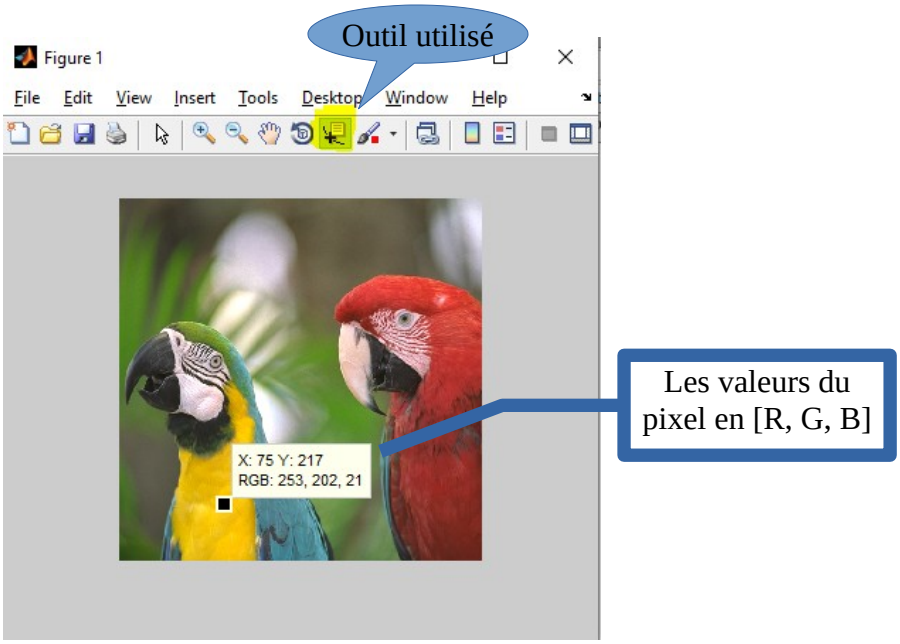
1- on ouvre l'image **Perroquet.tif** :

```
%% 1)
im = (imread('images/Perroquet.tif'));    figure(1), imshow(im);
```

on obtiens :

Figure 19: image Perroquet.tif

2- On affiche l'image avec la valeur des pixels pointé par la souris dans les trois bandes :



3- On determine les pixels de références et onles implémentent sur le code :

```
%%%3) Pixels de référenes: [R G B]
p1 = [250, 198, 0];      % jaune
p2 = [225, 77, 72];      % rouge
p3 = [87, 142, 152];     % bleu
p4 = [80 145 41];        % vert
```

Remarque :
le plus de classes qu'on détermine, le mieux la segmentation serai. On peut étayer cette remarque par l'image resultante apres déroulement du programme.

4- Application de la classification supervisée, en utilisant la distance euclidienne entre un **pixel$_{ij}$** et les pixels de références des différentes classes.

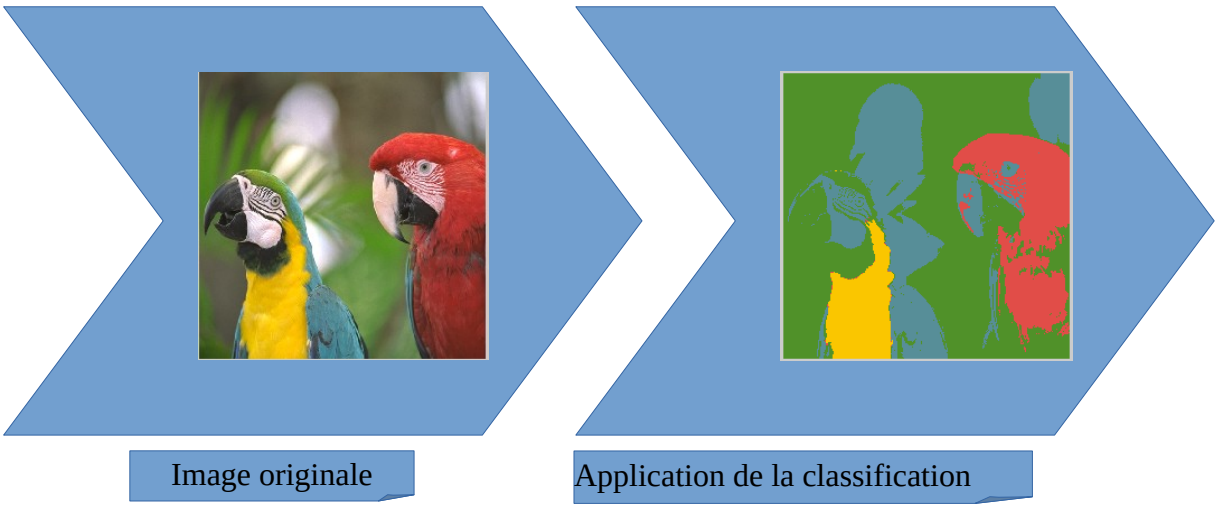
```
%%Extraire la taille de l'image pour faire le parcour de la boucle
[w, h, k] = size(im);
%%boucle qui parcour toute l'image:
for n= 1:h
   for m = 1:w
      p = [im(n, m, 1),   im(n, m, 2),    im(n, m, 3)]; %pixel couleur à la position (n, m)

      d1 =  distance_p(p1, p);  %distance entre pixel donné et le pixel de référene jaune.
      d2 =  distance_p(p2, p);  %distance entre pixel donné et le pixel de référene rouge.
      d3 =  distance_p(p3, p);  %distance entre pixel donné et le pixel de référene bleu.
      d4 =  distance_p(p4, p);  %distance entre pixel donné et le pixel de référene vert.

      if min([d1, d2, d3, d4]) == d1  %ou on utilise (d1 < d2 & d1 < d3 & d1 < d4).
          im(n, m, 1)=p1(1);    im(n, m, 2) = p1(2);    im(n, m, 3) = p1(3);
      elseif  min([d1, d2, d3, d4]) == d2
         im(n, m, 1)=p2(1);    im(n, m, 2) = p2(2);    im(n, m, 3) = p2(3);
      elseif  min([d1, d2, d3, d4]) == d4
         im(n, m, 1)=p4(1);    im(n, m, 2) = p4(2);    im(n, m, 3) = p4(3);
      elseif  min([d1, d2, d3, d4]) == d3
         im(n, m, 1)=p3(1);    im(n, m, 2) = p3(2);    im(n, m, 3) = p3(3);
      else
         im(n, m, 1)=255;    im(n, m, 2) = 255;    im(n, m, 3) = 255;
      end
   end
end
%%Affichage du résultat:
figure(2), imshow(im);
```

Figure 20: programme de classificartion supervisée.

Le résultat de l'affichage est comme suite :



Remarque :
on remarque que les 4 classes sont bien distinctes, et que la couleur verte est la plus dominante car elle est la plus proche de la majorité des pixels. Aussi, la distance euclidienne est un calcule facile et directe qui ne nécessite pas d'autre paramètres tel que la variance, la moyenne, l'écart-type…

Conclusion

À travers ce travail on a pu apprendre a mieux se familiariser avec l'environement matlab et ses fonction.
On a appris a appliquer des traitements d'images tel que la segmentation, la classification, et application de filtres. Aussi, le te travail dans plusieurs espaces de couleur, tel que RGB, HSV…etc.