

CYCLE INITIAL EN TECHNOLOGIES DE L'INFORMATION DE SAINT-ÉTIENNE

Travaux Pratiques

Traitement d'Images

EVA MATORANA- LUCAS LESCURE

Table of Content

1. Introduction	3
2. Traitement de l'image sous MATLAB	3
2.1. Prétraitement	4
2.2. Filtrage Gaussien.	4
2.3. Fermeture Morphologique de l'image	6
2.4. Segmentation et Étiquetage	6
2.5. Séparation des pièces	7
3. Annexe	8

1. Introduction

Dans ce TP, notre objectif était de concevoir et mettre en œuvre une solution pour compter et identifier efficacement des pièces, en prenant en compte les cas où les pièces se chevauchent. En utilisant une combinaison de techniques de traitement d'image, nous avons réussi à développer un système capable de faire face à ces cas pouvant être rencontré dans des applications pratiques réelles [Figure 1.1 - Figure 1.2].

Les techniques employées dans notre approche incluent le filtrage gaussien, la fermeture morphologique, la segmentation et l'étiquetage pour attribuer des identifiants uniques à chaque pièce. La combinaison de ces méthodes nous a permis de traiter l'image, et de discerner et d'étiqueter avec précision chaque pièce présente dans l'image.



Figure 1.1. Situation Idéale de distribution



Figure 1.2. Situation particulière réelle

2. Traitement de l'image sous MATLAB

Dans le but de détecter les différentes pièces contenues dans l'image on aboutira à une segmentation qui nous permettra de traiter chaque pièce séparément et de déterminer si elle se superpose, et dans ce cas refaire une segmentation adapté en distinguant les deux pièces superposés.

De plus, on vise à rendre notre code facilement malléable, nécessitant d'ajuster que 3 paramètres pour traiter l'image correctement:

- `correlationThreshold` : le seuil de tolérance pour la détermination de la circularité des pièces, permettant de déterminer quelles pièces se superposent.
- `closingOrder` : degré de fermeture de l'image pour remplir les parasites éventuels qui peuvent être formés suite à des reflets.
- `objectFilter` : taille maximale des objets à filtrer

2.1. Prétraitemet

Pour commencer avec le traitement de l'image on commence par convertir l'image en niveau de gris en utilisant la fonction `rgb2gray()` et on redimensionne la taille de l'image à une matrice 255x255 pour effectuer des traitements plus rapides avec la fonction `imresize()`.

```
imageFile = imread('g2.jpg');
grayImage= rgb2gray(imageFile);

grayImage = imresize(grayImage,[255,255]);
[sizeIX,sizeIY] = size(grayImage);
```



Figure 2.1. Image en niveau de gris

2.2. Filtrage Gaussien

Puisque l'on consacre notre projet entièrement à la détection d'objet on fera le choix de prendre un filtre Gaussien plutôt qu'un filtre moyenneur afin de préserver les contours des pièces. De plus la puissance de ce filtre peut-être ajustable ce qui permet un degré de liberté au niveau de la mise au point de l'image.

Pour effectuer ce type de filtrage on utilise la propriété dispersive de la distribution gaussienne afin d'appliquer d'appliquer les coefficients appropriés sur un noyau 3x3. On retrouve donc que la Gaussienne en 2 dimension pour notre noyau s'exprime :

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

On nomme donc les coefficients du noyaux comme étant :

```
c1 = (4 * pi * Strength ^ 2);
c2 = (8 * pi * Strength ^ 2);
c3 = (16 * pi * Strength ^ 2);
```

Cependant pour éviter d'éclaircir l'image en appliquant ce filtre pour des valeurs de `Strength` on veut que la somme de tout ces coefficients soient égaux à 1. On va donc effectuer une normalisation:

```
coef_sum = 4 / c3 + 4 / c2 + 1 / c1;
c1 = c1 / coef_sum;
c2 = c2 / coef_sum;
c3 = c3 / coef_sum;
```

Maintenant il ne manque plus qu'à trouver la valeur optimale pour le traitement d'image. En faisant varier le facteur de puissance `Strength` de 0.3 à 0.8 par pas de 0.05. On applique donc le filtre pour chaque valeur de `Strength`, et on effectue un seuillage optimal en utilisant la fonction `graythresh()` qui renvoie la valeur du seuil entre 0 et 1.

```
for i = 2:sizeIX-1
for j = 2:sizeIY-1
    imageAverage(i, j) = grayImage(i-1, j-1)/c3 + grayImage(i+1, j-1)/c3 + ...
    grayImage(i-1, j+1)/c3 + grayImage(i+1, j+1)/c3+ ...
    grayImage(i+1, j)/c2 + grayImage(i-1, j)/c2 + ...
```

```

        grayImage(i, j+1)/c2 + grayImage(i, j-1)/c2 + ...
        grayImage(i, j)/c1;
    end
end

```

Pour maintenant mesurer la qualité de l'image ainsi que la différence de qualité avant et après seuillage, on va s'intéresser à la fonction `edges()` qui nous permet de trouver les contours des objets de l'image. Dans le but d'avoir une image seuillée optimale, on fait la différence de ces deux contours pour pouvoir déterminer si les objets dans l'image ont été effacé ou pas lors de l'étape de seuillage.

```

first_edges = edge(grayImage, 'Canny');
Threshold = 255 * graythresh(imageAverage);

imageThreshold = uint8(255 * (imageAverage < Threshold));
second_edges = edge(imageThreshold, 'canny');
edge_diff = sum(abs((first_edges(:) - second_edges(:)))));

```

On conserve alors la valeur de `Strength` pour laquelle cette différence entre les contours est minimale dans la variable `best_strength`, assurant ainsi que le seuillage n'efface pas les objets de l'image filtré.

```

if edge_diff < best_edge_diff
    best_strength = averagingStrength;
    best_edge_diff = edge_diff;
end

```

Avec la valeur optimale de puissance pour le filtre Gaussien on effectue un dernier filtrage en prenant `best_strength` pour s'assurer que le seuillage de l'image conserve tout les objets dans l'image le résultat étant montré dans la figure ci-dessous.

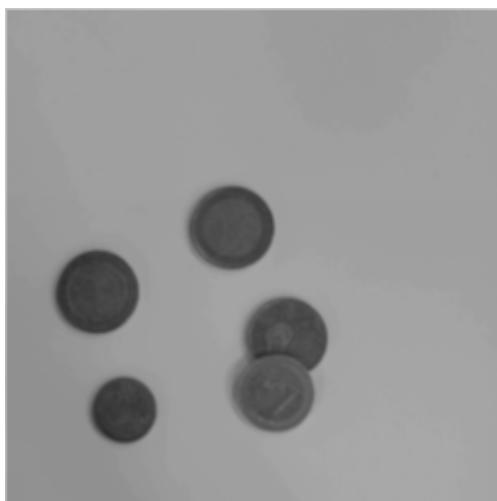


Figure 2.2. Image optimale du filtre Gaussien

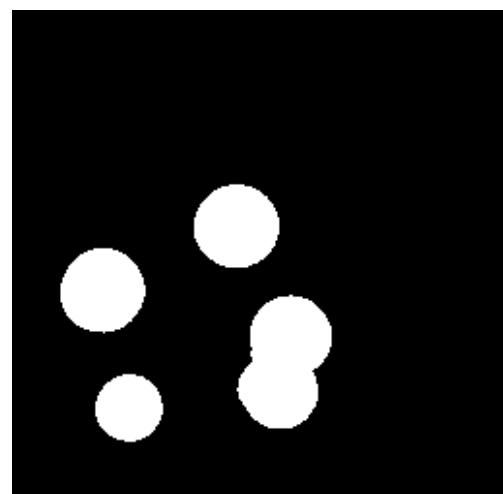


Figure 2.3. Image seuillée

2.3. Fermeture Morphologique de l'image

Dans le cas éventuel qu'il y a des parasites à l'intérieur de l'image on effectue une fermeture en applicant premièrement une dilatation puis une erosion de l'image. Pour pouvoir contrôler la puissance de la fermeture on nomme une variable `closingOrder` qui détermine le nombre de fois à répéter la fermeture.

```
% Dilatation de l'image
imageDilation = imageThreshold;
imageDilationOrder = imageThreshold;

for n = 1:closingOrder
    for i = 2:sizeIX-1
        for j = 2:sizeIY-1
            if(imageDilationOrder(i,j) == 0 )
                if(imageDilationOrder(i-1,j) == 255 || ...
                   imageDilationOrder(i+1,j) == 255 || ...
                   imageDilationOrder(i,j-1) == 255 || ...
                   imageDilationOrder(i,j+1) == 255)
                    imageDilation(i,j) = uint8(255);
            end
        end
    end
end
imageDilationOrder = imageDilation;
end

% Erosion de l'image

imageErosion = imageDilation;
imageErosionOrder = imageThreshold; % image recursive pour l'ordre de fermeture

for n = 1:closingOrder
    for i = 2:sizeIX-1
        for j = 2:sizeIY-1
            if(imageErosionOrder(i,j) == 0 )
                if(imageErosionOrder(i-1,j) == 255 || ...
                   imageErosionOrder(i+1,j) == 255 || ...
                   imageErosionOrder(i,j-1) == 255 || ...
                   imageErosionOrder(i,j+1) == 255)
                    imageErosion(i,j) = uint8(255);
            end
        end
    end
end
imageErosionOrder = imageErosion;
end

imageClosing = imageErosion;
```

2.4. Segmentation et Étiquetage

Après avoir réalisé la fermeture morphologique on fait une segmentation en 4 étapes puis on liste toutes les objets détectés en utilisant leur étiquette et on filtre les objets qui ne sont pas des pièces un applicant un seuil de taille `objectFilter`

```

objectFilter=200;
for k=1:254
    label = sum(sum(imageSegmented == k));
    if label > objectFilter
        listObjects(k)= label;
    end
end
obj_number = nnz(listObjects); %Nombre d' objets
labelObject = (find(listObjects)); %Etiquettes objets

```

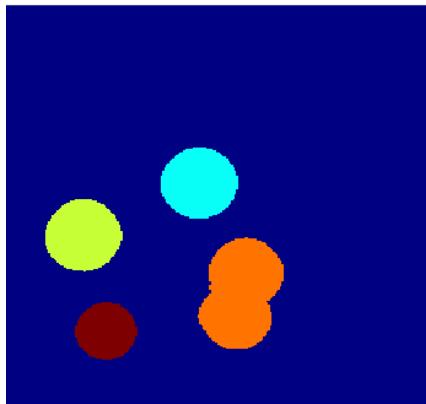


Figure 2.4. Image segmentée

Après avoir segmenté et étiquetter les objets il faut maintenant déterminer si certaines pièces sont l'une par dessus l'autre. Pour cela on se propose de mesurer la circularité de chaque objet en utilisant la fonction `regionprops()` qui permet de retourner des informations sur chaque objet.

Grâce à celle-ci on récupère les coordonnées du centre géométrique de chaque objet ainsi que leur circularité et leur aire. Dans le cas où l'objet n'est pas un disque parfait alors la circularité s'écarte de 1, en appliquant un seuil pour la tolérance de la circularité on distingue ainsi quels objets sont des pièces qui se superposent.

```

superpose = 0;
for k=1:obj_number
    areaObject(k) = sum(sum(imageSegmented == labelObject(k)));
    Objects(k) = regionprops((imageSegmented == labelObject(k)), ...
        "Circularity", "Centroid", "Area");
    Radius(k) = sqrt(Objects(k).Area) / pi
    if Objects(k).Circularity < correlationThreshold
        display(sprintf(['Il y a une superposition de deux pieces ...
            voir la piece : %i\n'], k))
        superpose = 1;
    end
end

```

2.5. Séparation des pièces

On cherche ensuite à séparer les pièces qui se superposent en utilisant la fonction `imfindcircles` qui permet d'identifier les cercles dans l'image et de retourner leur rayon ainsi que leur centre.

```

min_radius = uint8(min(Radius)) ;
max_radius = uint8(max(Radius)) + 10;
edges_img = edge(imageSegmented, 'canny');

[centers, radii] = imfindcircles(edges_img, [min_radius, max_radius], ...
    'ObjectPolarity', 'bright', 'Sensitivity', 0.9025);

```

Grâce à ces données on peut alors dessiner sur une nouvelle grille de taille 255x255 les pièces détectées en utilisant la fonction `meshgrid()` et en appliquant le mask de chaque pièce par dessus:

```
segmented_img = zeros( size( grayImage ) );

num_circles = size( centers , 1 );
for i = 1:num_circles
    center = centers(i, :);
    radius = radii(i);
    [X, Y] = meshgrid(1:size( grayImage , 2), 1:size( grayImage , 1 ));

    circle_mask = ((X - center(1)).^2 + (Y - center(2)).^2) <= radius.^2;

    segmented_img( circle_mask ) = i ;
end
```

On obtient alors l'image suivante :

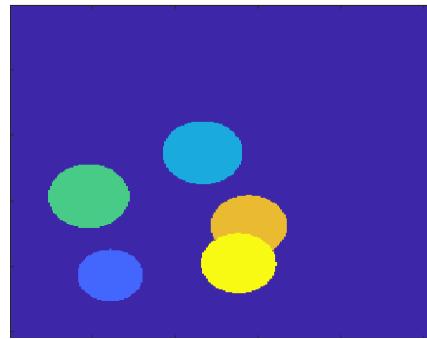


Figure 2.5. Séparation des pièces

On finit alors le traitement par ajouter du texte sur chaque pièce:

```
for k=1:num_circles
    text( centers(k,1) , centers(k,2) , [ 'Coin ' , num2str(k) ] , ...
        'HorizontalAlignment' , 'center' , 'FontSize' ,8 );
end
```

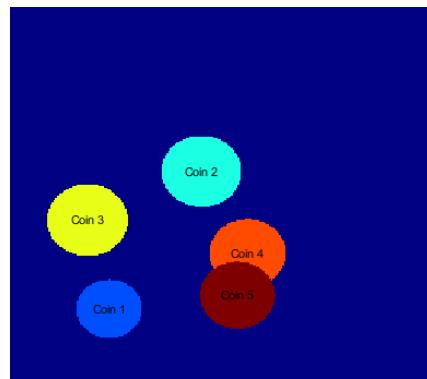


Figure 2.6. Pièces séparés et différentiés

3. Annexe

Code MATLAB (UTF-8):

```

clear all;
close all;

%% -- Valeurs a configurer --
correlationThreshold = 0.925 % Seuil de tolerance de la circularite
closingOrder = 1 % ordre de la fermeture configurable

%% -- Pretraitement
% Recuperation de l'image et adaptation de taille
imageFile = imread('g3.jpg');
grayImage= rgb2gray(imageFile);

grayImage = imresize(grayImage,[255,255]);
[sizeIX, sizeIY] = size(grayImage);

% Filtrage par filtre Gaussien 3x3
imageAverage = grayImage;

%% -- Filtrage Gaussien --
% Coefficients gaussiens

best_edge_diff = Inf;

for Strength = 0.35:0.05:0.8
c1 = (4*pi*Strength^2);
c2 = (8*pi*Strength^2);
c3 = (16*pi*Strength^2);

% Normalisation des coefficients
total_sum = 4 / c3 + 4 / c2 + 1 / c1;
c1 = c1 / total_sum;
c2 = c2 / total_sum;
c3 = c3 / total_sum;

for i = 2:sizeIX-1
    for j = 2:sizeIY-1
        imageAverage(i,j) = grayImage(i-1,j-1)/c3 + grayImage(i+1,j-1)/c3 + ...
            grayImage(i-1,+1)/c3 + grayImage(i+1,j+1)/c3+ ...
            grayImage(i+1,j)/c2 + grayImage(i-1,j)/c2 + ...
            grayImage(i,j+1)/c2 + grayImage(i,j-1)/c2 + ...
            grayImage(i,j)/c1;
    end
end

first_edges = edge(grayImage, 'Canny');
% Seuillage de l'image
Threshold = 255*graythresh(imageAverage);

imageThreshold = uint8(255*(imageAverage < Threshold));
second_edges = edge(imageThreshold, 'canny');
edge_diff= sum(abs((first_edges(:) - second_edges(:))));

if edge_diff < best_edge_diff

```

```
best_strength = Strength;
best_edge_diff = edge_diff;
end

end

c1 = (4*pi*best_strength^2);
c2 = (8*pi*best_strength^2);
c3 = (16*pi*best_strength^2);

% Normalisation des coefficients
total_sum = 4 / c3 + 4 / c2 + 1 / c1;
c1 = c1 / total_sum;
c2 = c2 / total_sum;
c3 = c3 / total_sum;

for i = 2:sizeIX-1
    for j = 2:sizeIY-1
        imageAverage(i,j) = grayImage(i-1,j-1)/c3 + grayImage(i+1,j-1)/c3 + ...
            grayImage(i-1,+1)/c3 + grayImage(i+1,j+1)/c3+ ...
            grayImage(i+1,j)/c2 + grayImage(i-1,j)/c2 + ...
            grayImage(i,j+1)/c2 + grayImage(i,j-1)/c2 + ...
            grayImage(i,j)/c1;
    end
end

imageAverage = uint8(imageAverage);

% Adaptation du seuillage par technique d'Otsu
Threshold = 255*graythresh(imageAverage);

% Seuillage de l'image
imageThreshold = uint8(255*(imageAverage<Threshold));

%% -- Fermeture de l'image --
% Dilatation de l'image
imageDilation = imageThreshold;
imageDilationOrder = imageThreshold; % image recursive pour l'ordre de dilatation

for n = 1:closingOrder
    for i = 2:sizeIX-1
        for j = 2:sizeIY-1
            if(imageDilationOrder(i,j) == 0 )
                if(imageDilationOrder(i-1,j) == 255 || ...
                    imageDilationOrder(i+1,j) == 255 || ...
                    imageDilationOrder(i,j-1) == 255 || ...
                    imageDilationOrder(i,j+1) == 255)
                    imageDilation(i,j) = uint8(255);
                end
            end
        end
    end
end
```

```

imageDilationOrder = imageDilation;
end

% Erosion de l'image

imageErosion = imageDilation;
imageErosionOrder = imageThreshold; % image recursive pour l'ordre de fermeture

for n = 1:closingOrder
    for i = 2:sizeIX-1
        for j = 2:sizeIY-1
            if(imageErosionOrder(i,j) == 0)
                if(imageErosionOrder(i-1,j) == 255 || ...
                   imageErosionOrder(i+1,j) == 255 || ...
                   imageErosionOrder(i,j-1) == 255 || ...
                   imageErosionOrder(i,j+1) == 255)
                    imageErosion(i,j) = uint8(255);
                end
            end
        end
    end
    imageErosionOrder = imageErosion;
end

imageClosing = imageErosion;

%% -- Segmentation --

imageSegmented = imageClosing;
label_init=60;
label = label_init;

% Premiere passe : depart en haut a gauche
for i = 2:sizeIX
    for j = 2:sizeIY
        if (imageSegmented(i,j) > 0)
            if (imageSegmented(i-1,j) == 0 && imageSegmented(i,j-1) == 0)
                imageSegmented(i,j) = label ;
                label = label + 2 ;
            else
                if (imageSegmented(i-1,j) > 0 && imageSegmented(i,j-1) > 0)
                    imageSegmented(i,j) = min([imageSegmented(i-1,j), ...
                                              imageSegmented(i,j-1)]) ;
                elseif (imageSegmented(i-1,j) > 0 && imageSegmented(i,j-1) == 0)
                    imageSegmented(i,j) = imageSegmented(i-1,j) ;
                else
                    imageSegmented(i,j) = imageSegmented(i,j-1) ;
                end
            end
        end
    end
end

```

```
end
```

```
% Deuxieme passe : depart en haut a droite
for i = 2:sizeIX
    for j = sizeIY-1:-1:1
        if (imageSegmented(i,j) > 0)
            if (imageSegmented(i-1,j) > 0 && imageSegmented(i,j+1) > 0)
                imageSegmented(i,j) = min([imageSegmented(i,j), ...
                    imageSegmented(i-1,j), imageSegmented(i,j+1)]);
            elseif (imageSegmented(i-1,j) > 0 && imageSegmented(i,j+1) == 0)
                imageSegmented(i,j) = min([imageSegmented(i,j), imageSegmented(i-1,j)]);
            elseif (imageSegmented(i-1,j) == 0 && imageSegmented(i,j+1) > 0)
                imageSegmented(i,j) = min([imageSegmented(i,j), imageSegmented(i,j+1)]);
            end
        end
    end
end
```

```
%Troisieme passe : depart en bas a droite
for i = sizeIX-1:-1:1
    for j = sizeIY-1:-1:1
        if (imageSegmented(i,j) > 0)
            if (imageSegmented(i+1,j) > 0 && imageSegmented(i,j+1) > 0)
                imageSegmented(i,j) = min([imageSegmented(i,j), ...
                    imageSegmented(i+1,j), imageSegmented(i,j+1)]);
            elseif (imageSegmented(i+1,j) > 0 && imageSegmented(i,j+1) == 0)
                imageSegmented(i,j) = min([imageSegmented(i,j), imageSegmented(i+1,j)]);
            elseif (imageSegmented(i+1,j) == 0 && imageSegmented(i,j+1) > 0)
                imageSegmented(i,j) = min([imageSegmented(i,j), imageSegmented(i,j+1)]);
            end
        end
    end
end
```

```
% Quatrieme passe : depart en bas a gauche
for i = sizeIX-1:-1:1
    for j = 2:sizeIY
        if (imageSegmented(i,j) > 0)
            if (imageSegmented(i+1,j) > 0 && imageSegmented(i,j-1) > 0)
                imageSegmented(i,j) = min([imageSegmented(i,j), ...
                    imageSegmented(i+1,j), imageSegmented(i,j-1)]);
            elseif (imageSegmented(i+1,j) > 0 && imageSegmented(i,j-1) == 0)
                imageSegmented(i,j) = min([imageSegmented(i,j), imageSegmented(i+1,j)]);
            elseif (imageSegmented(i-1,j) == 0 && imageSegmented(i,j-1) > 0)
                imageSegmented(i,j) = min([imageSegmented(i,j), imageSegmented(i,j-1)]);
            end
        end
    end
end
```

```
% On reunit dans une liste 1x254 les differents objets trouves
objectFilter = 200
for k=1:254
    label = sum(sum(imageSegmented == k));
    if label > objectFilter
        listObjects(k)= label;
    end
end

% Affichage du nombre d'objets listes
obj_number = nnz(listObjects);

% Etiquetage des objets en recuperant leur valeur de la liste
labelObject = (find(listObjects))

%Calcul de l'aire reelle et l'aire du contour de chaque objets

superpose = 0;
count = 0;
for k=1:obj_number
    areaObject(k) = sum(sum(imageSegmented == labelObject(k)));
    Objects(k) = regionprops((imageSegmented == labelObject(k)), ...
        "Circularity", "Centroid", "Area");
    Radius(k) = sqrt(Objects(k).Area) / pi ;
    if Objects(k).Circularity < correlationThreshold
        display(sprintf(['Il y a une superposition de deux pieces ...
            voir la piece : %i\n'] ,k+count));
        count = count +1;
        superpose = 1;
    end
end

min_radius = uint8(min(Radius)) ;
max_radius = uint8(max(Radius)) + 10;
edges_img = edge(imageSegmented, 'canny');

[centers, radii] = imfindcircles(edges_img, [min_radius, max_radius], ...
    'ObjectPolarity', 'bright', 'Sensitivity', 0.9025);

% Differentiation
segmented_img = zeros(size(grayImage));

num_circles = size(centers, 1);
for i = 1:num_circles
    center = centers(i, :);
    radius = radii(i);
    [X, Y] = meshgrid(1:size(grayImage, 2), 1:size(grayImage, 1));

    circle_mask = ((X - center(1)).^2 + (Y - center(2)).^2) <= radius.^2;
```

```

segmented_img(circle_mask) = i;
end

if superpose == 0
    disp('Aucune piece ne se superpose, le nombre de pieces est donc :')
    disp(num_circles)
else
    disp(sprintf(['Le nombre de pieces est : %i'], num_circles));
end

figure(3); subplot(1,2,1); imhist(imageAverage);
figure(3); subplot(1,2,2); imhist(grayImage);

figure(1); subplot(2,3,2); imshow(imageAverage); title('Averaged')
figure(1); subplot(2,3,1); imshow(grayImage); title('Grayed')
figure(1); subplot(2,3,3); imshow(imageThreshold); title('Thresholded')
figure(1); subplot(2,3,4); imshow(imageDilation); title('Dilated')
figure(1); subplot(2,3,5); imshow(imageClosing); title('Closing')
figure(1); subplot(2,3,6); imagesc(imageSegmented); axis off;
colormap('jet'); colorbar; title('Segmented');
figure(2); imagesc(segmented_img); axis off;
colormap('jet'); colorbar; title('Result');

sort_centers = sortrows(centers, 2);
for k=1:num_circles
    text(sort_centers(k,1), sort_centers(k,2), [ 'Coin ', num2str(k)], ...
        'HorizontalAlignment', 'center', 'FontSize', 8);
end

```



Figure 3.1. Image pré-traitement

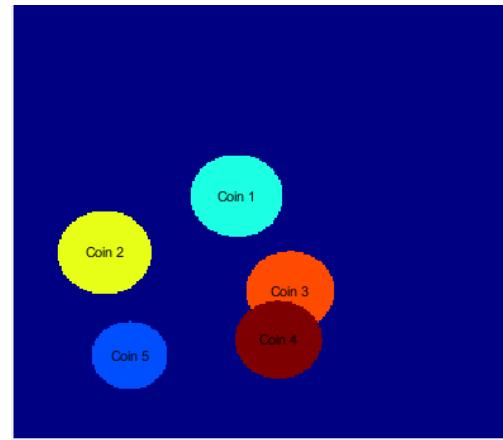


Figure 3.2. Image post-traitement



Figure 3.3. Image pré-traitement (avec ombres)

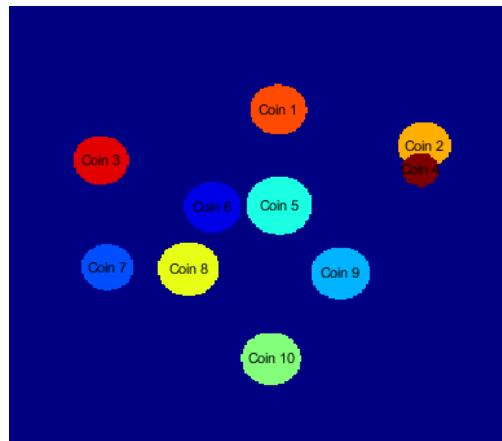


Figure 3.4. Image post-traitement



Figure 3.5. Image pré-traitement

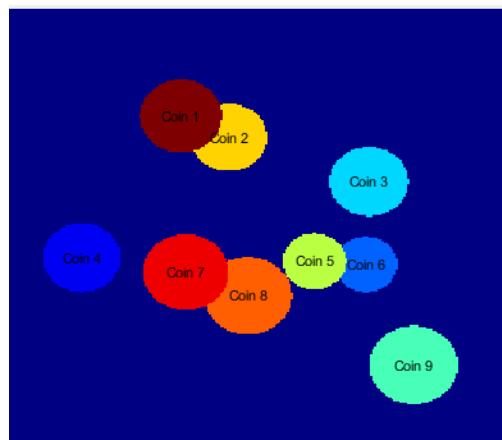


Figure 3.6. Image post-traitement