TS225

Compte rendu - Projet en Traitement des Images

Maxime PETERLIN - maxime.peterlin@enseirb-matmeca.fr
Gabriel VERMEULEN - gabriel@vermeulen.email

ENSEIRB-MATMECA, Bordeaux

1^{er} janvier 2014

Contents

1 Introduction **2**

2 Démarche algorithmique **2**

3 Implémentation **4**

4 Résultats **8**

5 Conclusion **13**

6 Annexes **14**

1 Introduction

Les codes barres permettent de représenter facilement une information alphanumérique sous forme d'image. On les retrouve au quotidien sous de mutliples formes. L'objectif de ce projet sera l'étude et le décodage de codes-barres unidimensionnels suivants la norme **EAN 13** nous permettant ainsi de nous familiariser avec les techniques fondamentales du traitement des images.

Premièrement, nous expliquerons notre démarche, ainsi que nos choix algorithmiques. Puis, nous passerons à l'impémentation. Enfin, nous étudierons les résultats obtenus à l'issue de ce projet.

2 Démarche algorithmique

Cette première section traite des algorithmes utilisés.

- **Détermination de la région d'intérêt**
  La première étape pour arriver au décodage des codes-barres a été la détection de leur zone d'intérêt. Afin de simplifier le traitement et la lecture du code-barres, on commence par passer l'image en niveau de gris en moyennant les composantes R, G et B de chaque pixel.

  Le principe de détection repose sur deux actions au préalable de l'utilisateur. En effet, il devra déterminer manuellement les bords droit et gauche du code-barres. Le programme se chargera ensuite de déterminer son extension horizontale.

  On appelle $A(x_1, y_1)$ et $B(x_2, y_2)$ les deux points déterminés par l'utilisateur. Grâce à ces derniers, on peut déterminer la région $\Re_0$ dont les coins supérieur gauche et inférieur droit ont pour coordonnées respectives (x_{min}, y_{min}) et (x_{max}, y_{max}). L'objectif, à présent, est d'étendre cette région verticalement, pour arriver à la région d'intérêt $\Re_T$.
  Pour ce faire, on part de la ligne de numéro y_{min} et on somme toutes les composantes de cette dernières ayant des abscisses comprises entre x_{min} et x_{max}. On va ensuite décrémenter la valeur du numéro de ligne y qui était initiallement à y_{min} et on réitère l'opération précédente sur la ligne actuelle. On compare alors les résultats obtenus et suivant la différence entre les lignes on considère ou non que l'on est sorti de la zone d'intérêt. On obtient finalement notre nouveau y_{min}.
  Le même raisonnement peut s'appliquer pour la recherche de la nouvelle valeur de y_{max}.

  Cette méthode est résumée par la formule mathématique suivante, où I est la matrice représentant l'image du code-barres et ϵ un paramètre de tolérance.

$$1 - \epsilon < \left| \frac{\sum_{x=x_{min}}^{x_{max}} I(x, y)}{\sum_{x=x_{min}}^{x_{max}} I(x, y_{min})} \right| < 1 + \epsilon$$

- **Estimation de la signature**
  La signature du code-barres est une représentation monodimensionnelle des informations qu'il porte. En effet, en théorie, il suffit d'avoir une seule ligne de pixels du code-barres pour pouvoir le décoder. Cependant, nous allons nous servir de cette redondance d'information pour pallier aux différentes erreurs que le code-barres pourrait comporter (rayures, flou, etc...).

On commence par projeter l'image sur l'axe horizontale. Cela se traduit par un moyennage des valeurs de chaque colonne comprise entre x_{min} et x_{max}.

Ensuite, on va binariser l'image pour qu'elle ne soit formée plus que par des pixels blancs ou noirs. Nous allons devoir trouver un seuil délimitant les valeurs qui nous intéressent et, pour ce faire, nous appliquerons l'algorithme de Otsu qui repose sur le calcul d'un histogramme h sur N. N sera le nombre de valeurs différentes prisent par les pixels de l'image.

On commence par calculer $w : k \mapsto w(k)$ et $\mu : k \mapsto \mu(k)$ avec $k \in [\![0, N-1]\!]$.
Notre seuil s est alors donné par $\mathrm{s} = \max_{k \in [\![0,N-1]\!]} w(k)[\mu(N-1) - \mu(k)]^2 + (1 - w(k))\mu(k)^2$
On peut, à présent, passer à l'étape de décision pour binariser l'image : 0 correspond à un pixel noir et 1 à un pixel blanc.

La prochaine étape est la suppression des bits inutiles sur les bords de l'image venant de la région d'intérêt sélectionnée approximativement par l'utilisateur. Maintenant que l'image a été binarisée, cette étape est simple, car il suffit de supprimer les pixels blancs les plus au bord de la signature s'ils existent.

- **Identification des chiffres**
  Nous savons qu'un code-barres est composé de 95 éléments de taille unitaire (i.e. les lignes le représentant). La première étape va être la séparation de la ligne obtenue précedemment en ces 95 élements. De plus, il va falloir identifier la couleur de chaque élément, car ces derniers sont composés de plusieurs pixels et il se peut que leur couleur ne soit pas uniforme, d'où la nécessité de prendre une décision sur cette dernière.

  Notre approche est relativement directe dans le sens où la couleur d'un élément donné sera celle des pixels de cette couleur qui sont majoritaires en nombre, ce qui nous permet de nous convertir un ensemble de pixels à un seul bit pour la description d'une barre.
  On arrive alors à ramener la signature du code-barres obtenue précedemment en un vecteur de 95 bits.

  L'étape suivante est la reconnaissance des chiffres. La signature d'un chiffre est donné par la concaténation de 7 éléments unitaires du code-barres. Dans le vecteur obtenu précédemment, les chiffres sont compris entre les indices 4 et 45, ainsi qu'entre les indices 51 et 92.

  Pour identifier les chiffres, nous utiliserons une matrice de 30 lignes et 7 colonnes comportant les signatures théoriques des chiffres. Nous comparons alors la signature observée à la matrice des signatures théoriques pour obtenir la valeur du chiffre encodé.

  La critère utilisé est le suivant.

$$c(s_{th}, s_p) = \frac{< s_{th} - \overline{s_{th}}, s_p - \overline{s_p} >}{\|s_{th} - \overline{s_{th}}\| \cdot \|s_p - \overline{s_p}\|}$$

  La signature retenue sera celle maximisant ce critère.

- **Cas des codes-barres *orientés***
  Les codes-barres étudiés n'auront pas toujours leurs lignes parallèles aux bords de l'image. Il va donc

falloir trouver un angle pour effectuer une rotation sur l'image afin qu'elle devienne *droite*.

Pour se faire, il faudra détecter les lignes grâce à la transformée de Hough, puis trouver leur orientation afin de trouver un angle de rotation.

3 Implémentation

Maintenant que nous avons expliqué les différents algorithmes que nous utiliserons pour décoder des codes-barres, nous allons passer à l'implémentation de ces derniers à l'aide de Matlab.

- **Passage en niveau de gris et rotation de l'image**
  La première fonction appelée par le programme est *init_code_barre* prennant en paramètre l'image sous forme matricielle et renvoyant l'image en niveau de gris et éventuellement tournée.

  On commence par récupérer la taille de l'image avec *size* pour vérifier si l'image n'est pas déjà en niveau de gris. Si elle ne l'est pas, on moyenne les trois composantes R, G et B, avant de les diviser par 255, sinon, on divise seulement par 255 pour se ramener à des valeurs entre 0 et 1.

  L'étape suivante est la rotation de l'image si l'on détecte que les lignes ne sont pas parallèles au bord de l'image. On utilise tout d'abord la fonction *edge* qui va mettre en évidence les lignes du code-barres. On peut ensuite passer à la transformée de Hough avec la fonction *hough* qui va nous renvoyer la matrice de la transformée H, ainsi que les angles θ associés.
  Il ne reste plus qu'à détecter l'orientation des lignes en utilisant la fonction *houghpeaks* localisant les pics de la transformée de Hough. L'indice de l'angle dans la matrice des θ sera donné par le maximum de la première colonne de la matrice renvoyée par *houghpeaks*. Il suffit alors de tourner l'image avec *imrotate* pour obtenir le résultat attendu.

- **Détection de la zone d'intérêt et estimation de la signature de l'image**

  La seconde fonction appelée sera *do_work* qui sera la routine principale utilisée pour le décodage.

  On commence par demander à l'utilisateur de choisir deux points se situant à gauche et à droite du code-barres grâce à la fonction *ginput* renvoyant deux vecteurs g_x et g_y.
  Ces vecteurs, en plus de l'image et d'un paramètre ϵ sont envoyés à la fonction *get_codes_barres_ligne* qui va détecter la zone d'intérêt et estimer la signature de l'image.

  Pour détecter la zone d'intérêt, on commence par définir les variables x_{min}, x_{max}, y_{min} et y_{max} à partir des vecteurs g_x et g_y.
  On définit également la variable r_{min} qui sera égale à la somme des valeurs des pixels de la ligne d'indice y_{min} compris entre les indices x_{min} et x_{max}. On définit r_{max} de la même manière pour la ligne d'indice y_{max}.
  On implémente alors l'algorithme vu dans la partie précédente lié à la détection de la signature à l'aide d'une boucle *for* et on sort de cette dernière dès que que la contrainte est remplie.

```
r_min = sum(img_ng(y_min, x_min:x_max));
for y=y_min:-1:1  % Calcul de la nouvelle valeur de y_min
    r = abs(sum(img_ng(y, x_min:x_max))/r_min);
    if r > 1+epsilon || r < 1-epsilon
        y_min = y;
        break;
```

```
    end
end

r_max = sum(img_ng(y_max, x_min:x_max));
for y=y_max:1:size_Y  % Calcul de la nouvelle valeur de y_max
    r = abs(sum(img_ng(y, x_min:x_max))/r_max);
    if r > 1+epsilon || r < 1-epsilon
        y_max = y;
        break;
    end
end
```

- **Binarisation de l'image et estimation de la signature**
  Maintenant que nous avons la zone d'intérêt, nous allons extraire de l'image sa signature.
  On va commencer par compter le nombre de valeurs uniques N comprises dans la matrice représentant la zone d'intérêt grâce aux fonctions *unique* et *length*. Ce qui nous permet de récuperer le nombre de classes sur lesquelles on appliquera la fonction *hist* qui nous renverra un histogramme lié à ces dernières.

  Nous allons à présent passer au calcul du critère d'Otsu qui va nous permettre de binariser l'image pour faciliter les traitements qui seront fait par la suite.
  On crée deux vecteurs w, mu et $crit$ de taille N dans lesquels seront stockés les valeurs calculées. Le seuil sera l'indice de la valeur maximale du vecteur $crit$ que l'on divisera par N.

  Pour obtenir la signature, il suffit d'utiliser la fonction *mean* que l'on appliquera sur la première dimension (i.e. les colonnes). Le résultat est un vecteur ligne qui comporte éventuellement des bits inutiles au début et à la fin de ce dernier. On considère que si les premiers ou derniers bits sont blancs alors ils sont inutiles et doivent être supprimés.

```
% Inversion des couleurs de l'image pour faciliter le traitement
code_barre_ligne = abs(code_barre_ligne - 1);

% Recherche de l'occurence du premier pixel blanc
% (qui est un pixel noir sur l'image originale)
x_min = 1;
if code_barre_ligne(x_min) == 0
    x_min = find(code_barre_ligne,1,'first');
end

% Recherche de l'occurence du dernier pixel blanc
x_max = length(code_barre_ligne);
if code_barre_ligne(x_max) == 0
    x_max = find(code_barre_ligne,1,'last');
end

% Extraction de la nouvelle signature
code_barre_ligne = code_barre_ligne(x_min:x_max);

% Inversion des couleurs pour se ramener aux couleurs d'origines
code_barre_ligne = abs(code_barre_ligne - 1);
```

- **Identification des chiffres**
  Nous savons qu'un code-barres est composé de 95 éléments. Il faut donc séparer l'image en ces différents éléments. Pour se faire, on divise la longueur de l'image par 95 ce qui nous donne un pas d'itération qui sera très certainement un nombre décimal.

```
step = length(code_barre_line_nb) / nb_elem;
```

On parcourt l'image avec des indices de début et de fin *last* et *next* propre à chaque élément et dépendant de la valeur entière du pas. On somme alors les valeurs des pixels compris entre ces indices et on regarde quelle est la valeur majoritaire en comparant par rapport à 0.5 (les valeurs étant uniquement des 0 ou des 1).

```
for j=1:nb_elem
    last = fix(i);
    next = fix(i+step);
    code_barre_code(j) = sum(code_barre_line_nb(last:next-1))/step >= 0.5;
    i = i + step;
end
```

Maintenant que nous avons déterminé complètements les éléments, il ne reste plus qu'à identifier les chiffres en utilisant la formule donnée dans la partie traitant des algorithmes.
On va dans un premier temps stocker les encodages de tous les chiffres dans une matrice de 30 lignes et 7 colonnes : les éléments blancs seront représentés par des 1 et les élements noirs par des 0.
Ensuite, on stocke dans un vecteur *chiffres_codes* les éléments correspondants aux barres du code-barres.

```
chiffres_codes = zeros(12, 7);

for i=1:6
    chiffres_codes(i, :) = code_barre_code(4+(i-1)*7:3+i*7);
    chiffres_codes(i+6, :) = code_barre_code(4+7*6+5+(i-1)*7:3+7*6+5+i*7);
end
```

A présent, nous allons passer à la détection de chaque chiffre.

```
% On parcourt les chiffres de 2 à 13, le premier sera calculé ultérieurement
for i=2:13
    % On regarde si le chiffre peut être décodé directement en regardant
    % s'il est présent dans la matrice "codes" qui contient les signatures
    % des chiffres encodés
    [~,indx] = ismember(chiffres_codes(i-1, :),codes,'rows');

    % S'il n'est pas présent, on recherche celui lui correspondant le plus
    % dans la matrice code
    if indx == 0
        % La matrice codes à laquelle on soustrait la moyenne de chaque ligne
        mat_mean_codes = codes-kron(mean(codes, 2),ones(1,7));

        % Calcul de la norme de chaque vecteur ligne de la matrice
        mat_norm_codes = arrayfun(@(idx) norm(mat_mean_codes(idx,:)), 1:size(mat_
```

```
        % Moyenne du vecteur représentant le chiffre qu'on veut identifier
        vect_mean = fliplr(chiffres_codes(i-1, :) - mean(chiffres_codes(i-1, :)))

        % Norme du vecteur représentant le chiffre qu'on veut identifier
        vect_norm = norm(vect_mean);
        [~, indx] = max( (mat_mean_codes*vect_mean)./mat_norm_codes )

        % Calcul de la valeur du chiffre en fonction de sa position
        % dans la matrice
                                                chiffres(i) = mod(indx-1, 10)
        verif = 1;
        if i <= 7
            % Enregistrement de la classe du chiffre pour les 7 premiers
            % chiffres identifiés
            premier_chiffre(i-1) = fix((indx-1)/10);
        end
    else
        if i <= 7

            premier_chiffre(i-1) = fix((indx-1)/10);
        end
        chiffres(i) = mod(indx-1, 10);
    end
end
```

Il ne reste plus qu'à calculer le premier chiffre et vérifier si le code est valide.
Pour calculer le premier chiffre, on crée une matrice *premier_codes* qui contiendra les différentes séquences de classes d'élément permettant d'identifier le premier chiffre. On utilise *ismember* pour récupérer l'indice de la séquence contenue dans *premier_chiffre* et on en déduit alors le premier chiffre.

Pour vérifier si le code est valide, on calcule une clé grâce à l'alorithme donné par la norme et on vérifie que la somme de cette clé avec le $13^{\text{è}}$ chiffre est un multiple de 10. Si c'est le cas, on passe la variable *verif* à 1 pour signifier que le code-barres est valide (on laisse la variable à 0 sinon).

```
cle = 0
for i=1:2:12
    chiffres(i)
    cle = cle + chiffres(i) + 3*chiffres(i+1);
end
chiffres(13);
verif = 0;
if mod(cle+chiffres(13),10) == 0
    verif = 1;
end
```

4 Résultats

Dans cette partie nous exposerons les résultats obtenus sur plusieurs codes-barres suite à notre implémentation des différents algorithmes énoncés supra.
Le premier exemple sera plus détaillé que les autres pour mettre en évidence l'enchaînement des différentes étapes.

- **Premier exemple**
  Le premier code-barres décodé sera le suivant.

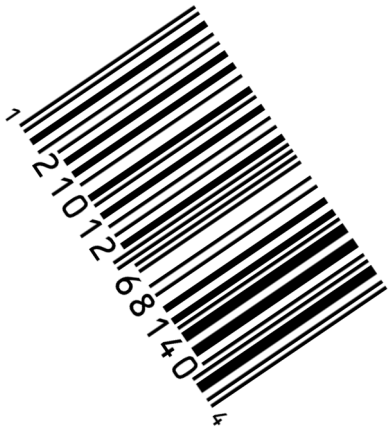

Figure 1: Code-barres n°1 orienté

La première étape est la rotation de l'image pour qu'elle redevienne droite en lui appliquant la transformée de Hough.

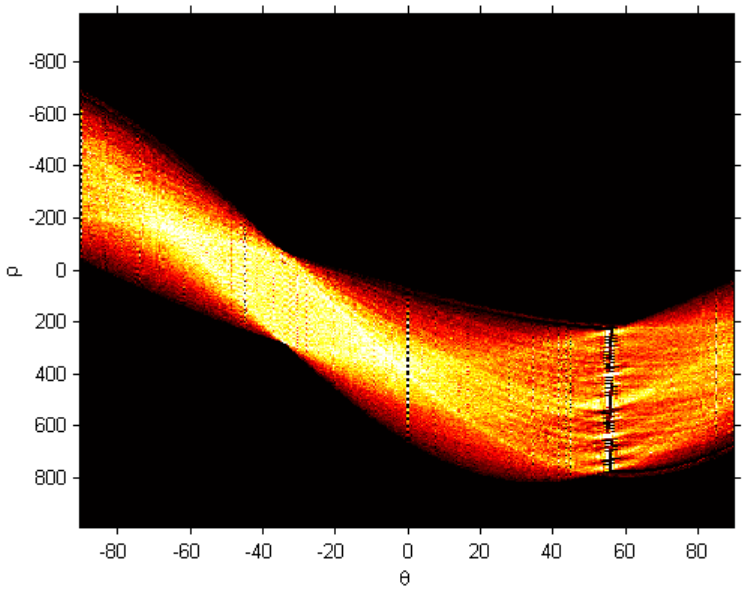

Figure 2: Représentation de la matrice de la transformée de Hough

On obtient alors le résultat suivant.

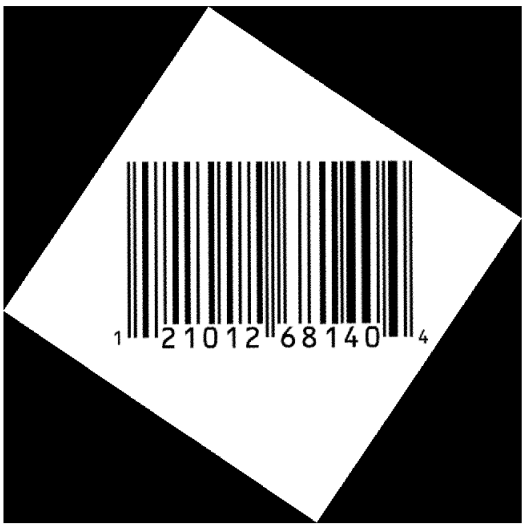

Figure 3: Image réorienté

On demande à présent à l'utilisateur d'identifier grossièrement la zone d'intérêt et le script va se charger de la délimiter précisément.

Figure 4: Zone d'intérêt

Maintenant que nous avons la zone d'intérêt, nous allons évaluer la signature de l'image, puis binariser cette dernière.

Il ne reste plus qu'à évaluer les chiffres et vérifier si le code-barres est valide.

Figure 5: Evaluation de la signature à partir de la zone d'intérêt

Figure 6: Binarisation de la signature

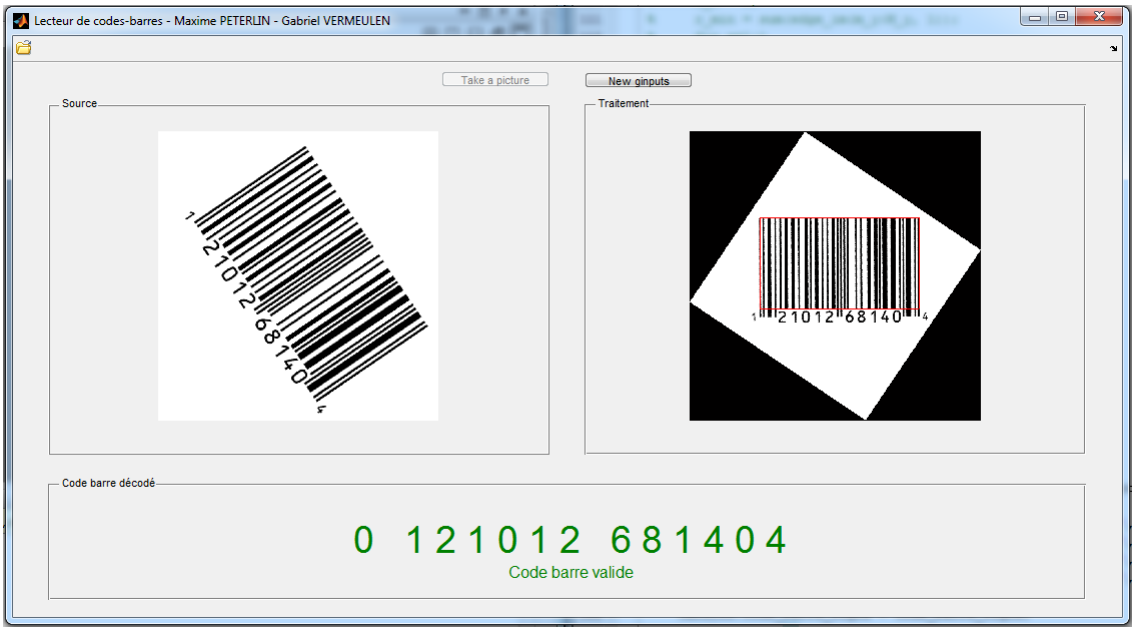

Figure 7: Résultat final

Le code-barres est bien décodé et valide.

- **Deuxième exemple** Nous avons essayé de tester la résistance de notre algorithme sur un code-barres flouté.

Figure 8: Code-barres n°2 - Sans flou

Sans aucun flou, le code-barres est reconnu et décodé sans aucun problème.

Figure 9: Code-barres n°2 - Flou léger

Avec un flou relativement faible, on remarque que le premier chiffre ne peut être reconnu, car un des chiffres a été identifié comme faisant parti de la classe C (rendant également le code invalide). Le code en lui-même est décodé cependant.

Figure 10: Code-barres n°2 - Flou fort

Par contre, lorsque le flou devient trop fort, on ne peut plus décoder correctement les chiffres.

- **Troisième exemple** A présent nous allons voir la résistance de l'algorithme face aux codes-barres âbimés.
  Nous nous basons sur le même code-barres que celui de l'exemple précédent.
  Selon la zone d'intérêt définie par l'utilisateur, le code-barres peut être décodé ou non.

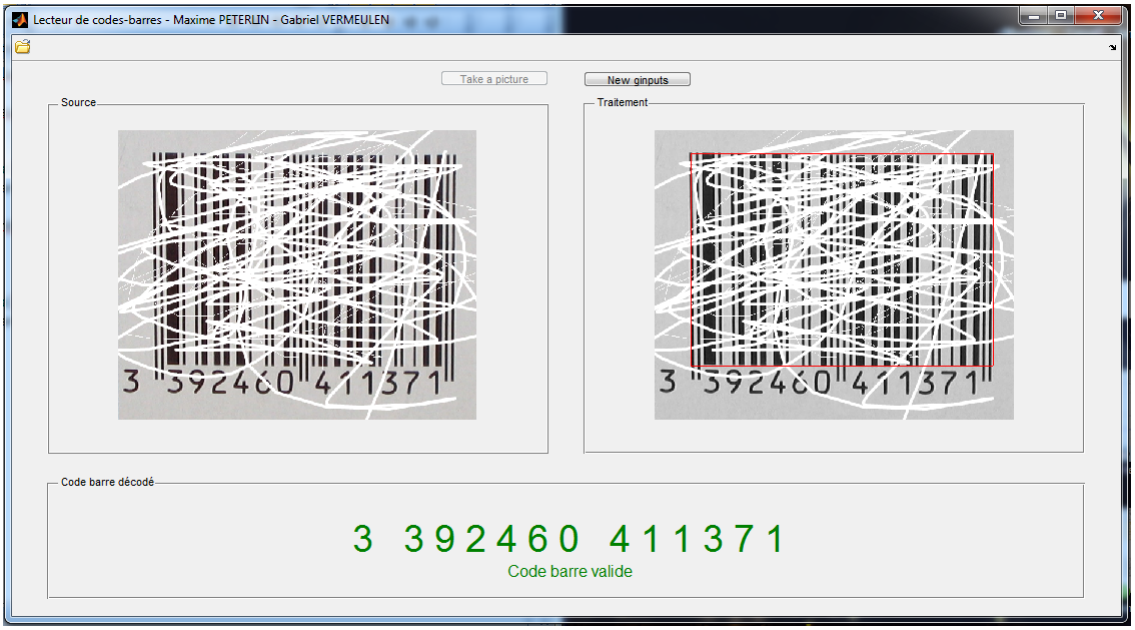

Figure 11: Code-barres n°3

Figure 12: Code-barres n°3

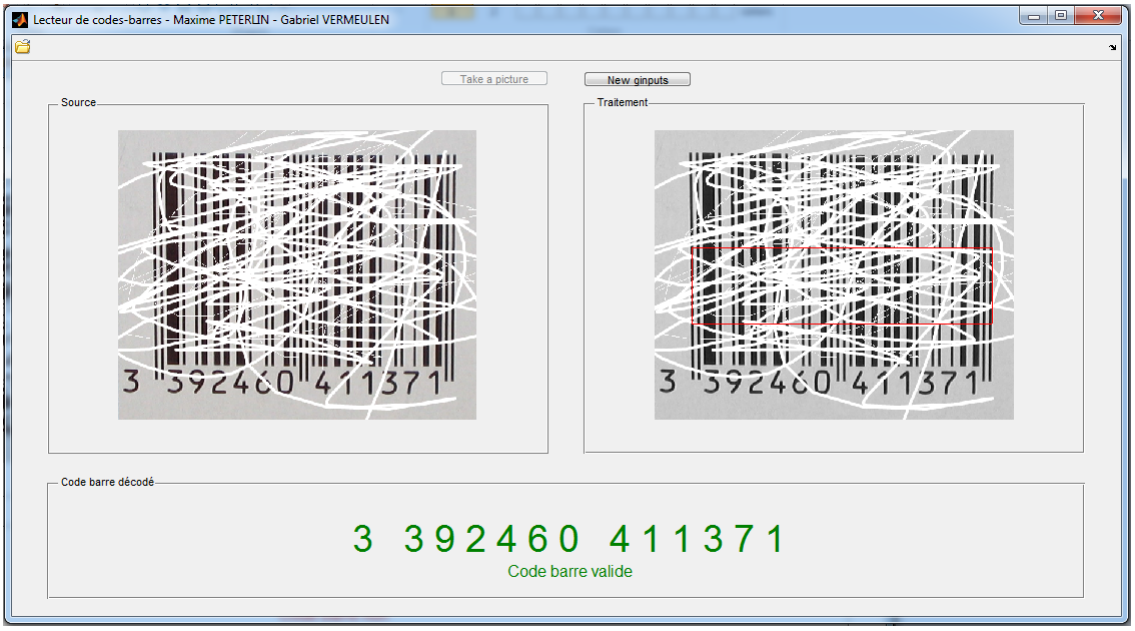

Figure 13: Code-barres n°3

5 Conclusion

En se basant sur les résultats de la partie précédente, on remarque que les résultats obtenus sont corrects, mais uniquement dans des situations bien précises. En effet, même si l'on arrive à décoder un code-barres représenté par une image de bonne qualité, s'il est trop flou ou bien trop abimé, il est souvent difficile d'obtenir le résultat souhaité. D'autant qu'il y a toujours le problème lié à l'interaction de l'utilisateur pour sélectionner la zone d'intérêt.
Ce projet nous aura cependant permis d'apprendre de nombreuses techniques et méthodes d'analyse liées au traitement des images (pour sélectionner la zone d'intérêt ou bien pour estimer la signature du code-barres, par exemple).

6 Bilan de l'organisation

- **Première séance :** Détermination de la zone d'intérêt et estimation de la signature du code-barres.
- **Entre la première et la deuxième séance (3h) :** Implémentation de l'algorithme permettant de décoder le code-barres.
- **Deuxième séance :** Mise en place de la GUI et implémentation de l'algorithme de rotation des images.
- **Entre la deuxième et la troisième séance (2h):** Fin de la mise en place de la GUI.
- **Troisième séance à la dernière :** Réglages de bugs et tentatives d'optimisation de l'algorithme.
- **Après la dernière séance :** Écriture du rapport.

7 Annexes

```
function [ img_out ] = init_code_barre( img_in )

    [~, ~, z] = size(img_in);

    if z == 3
        R = double(img_in(:,:,1));
        G = double(img_in(:,:,2));
        B = double(img_in(:,:,3));
        img_nb = (R+G+B)/3/255;
    else
        img_nb = double(img_in/255);
    end

    img_edge = edge(img_nb);
    [H,T,~] = hough(img_edge);
    peak = houghpeaks(H, 30);
    [m, r] = max(peak(:, 1));
    if m ~= 1
        angle = T(peak(r,2));
        img_out = imrotate(img_nb, angle);
    else
        img_out = img_nb;
    end

end

function do_work(hObject, handles)

    axes(handles.axes2);
    cla(handles.axes2);
    imshow(handles.img);
```

```
    [gx, gy] = get_ginput(handles);

    [ code_barre_ligne, x_min, x_max, y_min, y_max ] = get_code_barre_ligne( handles.i

    line([x_min+gx(1), x_max+gx(1)], [y_min, y_min], 'Color', 'red');
    line([x_min+gx(1), x_max+gx(1)], [y_max, y_max], 'Color', 'red');
    line([x_min+gx(1), x_min+gx(1)], [y_min, y_max], 'Color', 'red');
    line([x_max+gx(1), x_max+gx(1)], [y_min, y_max], 'Color', 'red');

    handles.code_barre_ligne = code_barre_ligne;
    handles.gx = gx;
    handles.gy = gy;

    guidata(hObject, handles);

                [ chiffres, verif ] = methode1( code_barre_ligne );
                set(handles.text_code, 'String', sprintf('%d   %d %d %d %d %d %d   %d
                if(verif == 1)
                                set(handles.text_code, 'ForegroundColor',[0 0.5 0]);
                                set(handles.text_verif, 'ForegroundColor',[0 0.5 0]);
                                set(handles.text_verif, 'String', 'Code barre valide'
                else
                                set(handles.text_code, 'ForegroundColor',[1 0 0]);
                                set(handles.text_verif, 'ForegroundColor',[1 0 0]);
                                set(handles.text_verif, 'String', 'Code barre non val
                end

end

function [ code_barre_ligne, x_min, x_max, y_min, y_max ] = get_code_barre_ligne( img

    x_min = fix(min(gx));
    x_max = fix(max(gx));
    y_min = fix(min(gy));
    y_max = fix(max(gy));

    [size_Y, ~] = size(img_ng);

    r_min = sum(img_ng(y_min, x_min:x_max));
    for y=y_min:-1:1
        r = abs(sum(img_ng(y, x_min:x_max))/r_min);
        if r > 1+epsilon || r < 1-epsilon
            y_min = y;
            break;
        end
    end

    r_max = sum(img_ng(y_max, x_min:x_max));
    for y=y_max:1:size_Y
        r = abs(sum(img_ng(y, x_min:x_max))/r_max);
```

```
        if r > 1+epsilon || r < 1-epsilon
            y_max = y;
            break;
        end
    end

    code_barre = img_ng(y_min:y_max, x_min:x_max);
    N = length(unique(code_barre));
    h = hist(code_barre, N);
    s = sum(h,2);
    h_sum = sum(s);

    w = zeros(N, 1);
    mu = zeros(N, 1);

    for k=1:N
        e = 0;
        for i=1:k
            e = e + i*s(i);
        end
        w(k) = sum(s(1:k))/h_sum;
        mu(k) = e/h_sum;
    end

    crit = zeros(N, 1);
    for k=1:N
        crit(k) = w(k)*(mu(N)-mu(k)).^2+(1-w(k))*mu(k).^2;
    end

    [~,i] = max(crit);
    seuil = i/N;

    code_barre_ligne = mean(code_barre, 1);
    code_barre_ligne = code_barre_ligne >= seuil;
    code_barre_ligne = abs(code_barre_ligne - 1);

    x_min = 1;
    if code_barre_ligne(x_min) == 0
        x_min = find(code_barre_ligne,1,'first');
    end

    x_max = length(code_barre_ligne);
    if code_barre_ligne(x_max) == 0
        x_max = find(code_barre_ligne,1,'last');
    end

    code_barre_ligne = code_barre_ligne(x_min:x_max);
    code_barre_ligne = abs(code_barre_ligne - 1);
end

function [ chiffres, verif ] = methode1( code_barre_line_nb )
```

```
nb_elem = 7*12+3*2+5;
code_barre_code = zeros(1, nb_elem);
step = length(code_barre_line_nb) / nb_elem;
i = 1;

for j=1:nb_elem
    last = fix(i);
    next = fix(i+step);
    code_barre_code(j) = sum(code_barre_line_nb(last:next-1))/step >= 0.5;
    i = i + step;
end

codes = [
    1,1,1,0,0,1,0;
    1,1,0,0,1,1,0;
    1,1,0,1,1,0,0;
    1,0,0,0,0,1,0;
    1,0,1,1,1,0,0;
    1,0,0,1,1,1,0;
    1,0,1,0,0,0,0;
    1,0,0,0,1,0,0;
    1,0,0,1,0,0,0;
    1,1,1,0,1,0,0;

    1,0,1,1,0,0,0;
    1,0,0,1,1,0,0;
    1,1,0,0,1,0,0;
    1,0,1,1,1,1,0;
    1,1,0,0,0,1,0;
    1,0,0,0,1,1,0;
    1,1,1,1,0,1,0;
    1,1,0,1,1,1,0;
    1,1,1,0,1,1,0;
    1,1,0,1,0,0,0;

    0,0,0,1,1,0,1;
    0,0,1,1,0,0,1;
    0,0,1,0,0,1,1;
    0,1,1,1,1,0,1;
    0,1,0,0,0,1,1;
    0,1,1,0,0,0,1;
    0,1,0,1,1,1,1;
    0,1,1,1,0,1,1;
    0,1,1,0,1,1,1;
    0,0,0,1,0,1,1;];

chiffres_codes = zeros(12, 7);

for i=1:6
```

```
    chiffres_codes(i, :) = code_barre_code(4+(i-1)*7:3+i*7);
    chiffres_codes(i+6, :) = code_barre_code(4+7*6+5+(i-1)*7:3+7*6+5+i*7);
end

chiffres = zeros(1, 13);
verif = 1;

premier_chiffre = zeros(1, 6);

for i=2:13
    [~,indx] = ismember(chiffres_codes(i-1, :),codes,'rows');

    if indx == 0
        mat_mean_codes = codes-kron(mean(codes, 2),ones(1,7));
        mat_norm_codes = arrayfun(@(idx) norm(mat_mean_codes(idx,:)), 1:size(mat_
        vect_mean = fliplr(chiffres_codes(i-1, :) - mean(chiffres_codes(i-1, :)))
        vect_norm = norm(vect_mean);
        [~, indx] = max( (mat_mean_codes*vect_mean)./mat_norm_codes )
        chiffres(i) = mod(indx-1, 10)
        verif = 1;
        if i <= 7
            premier_chiffre(i-1) = fix((indx-1)/10);
        end
    else
        chiffres(i) = mod(indx-1, 10);
    end

    if i <= 7
        premier_chiffre(i-1) = fix((indx-1)/10);
    end
end

premier_codes = [
    0 0 0 0 0 0;
    0 0 1 0 1 1;
    0 0 1 1 0 1;
    0 0 1 1 1 0;
    0 1 0 0 1 1;
    0 1 1 0 0 1;
    0 1 1 1 0 0;
    0 1 0 1 0 1;
    0 1 0 1 1 0;
    0 1 1 0 1 0;
   ];
[~,indx] = ismember(premier_chiffre,premier_codes,'rows');
chiffres(1) = indx-1;

cle = 0;
for i=1:2:12
    cle = cle + chiffres(i) + 3*chiffres(i+1);
```

```
    end

    verif = 0;
    if mod(cle,10) == 10-chiffres(13)
        verif = 1;
    end
end
```