TS 225 Projet Image

FRANCHI Loïc
BONDU Jimmy

Table des matières

1 Introduction

Les applications concernant les images ou la vidéo sont nombreuses, mais elles sont aussi complexes à mettre en oeuvre. C'est pourquoi, l'objectif de ce projet était de mettre en pratique les enseignements fondamentaux de traitements des images vus au cours du semestre. Pour cela, le travail à réaliser était de lire un code-barres à partir d'images numériques. Dans un premier temps, les codes-barres considérés étaient des photographies sur lesquelles les bords sont parallèles aux barres du code. Puis dans un second temps, ils pouvaient être orientés. Par ailleurs pour répondre à ces problématiques, l'intervention d'un opérateur a été considérée afin de délimiter la région de l'image contenant le code-barres.

2 Déroulement du projet

2.1 Cas d'un code-barres non orienté

2.1.1 Détermination de la région d'intérêt

Lors de cette première étape, les morceaux de l'image présents au-dessus et en dessous du codes-barres sont éliminés. L'opérateur avec sa souris trace une ligne quasi-horizontale traversant, au plus près, de part en part, le code-barres dans sa largeur. Cette ligne tracée renvoie deux coordonnées de deux points, qui sont : (x_1, y_1) et (x_2, y_2). Elles permettent de définir un rectangle dont les coins supérieur gauche et inférieur droit ont respectivement pour coordonnées : (x_{min}, y_{min}) et (x_{max}, y_{max}) avec $x_{min} = min(x_1, x_2)$, $y_{min} = min(y_1, y_2)$, $x_{max} = max(x_1, x_2)$ et $y_{max} = max(y_1, y_2)$. Ce rectangle est à l'origine de la région d'intérêt finale, notée $\Re_T$.

En effet, pour déterminer les ordonnées/la hauteur de $\Re_T$, la formule suivante calcule pour chaque ligne, la somme des pixels d'une même ligne et en fait le rapport avec la ligne correspondante au bord supérieur du rectangle $\Re_T$:

$$1 - \epsilon < \left| \frac{\sum_{x=x_{min}}^{x_{max}} I(x, y)}{\sum_{x=x_{min}}^{x_{max}} I(x, y_{min})} \right| < 1 + \epsilon \text{ avec } \epsilon \text{ un paramètre d'erreur ou de tolérance} \quad (1)$$

Si le résultat de ce rapport se retrouve entre les bornes définies i.e s'il est proche de 1, cela signifie que la ligne scrutée est constituée à priori des mêmes pixels que celle de la frontière haute de $\Re_T$. Ce qui sous-entend alors que ces lignes sont semblables. Il advient que plus le facteur ϵ est faible, plus la similitude entre la ligne de référence et les autres doit être importante. Ceci a pour conséquence que si une partie du code-barres est dégradée, la région d'intérêt s'arrêtera avant de prendre en considération une ligne de pixels composant ce secteur abîmé.

Avec la formule (1), la limite haute de la région d'intérêt a été déterminée. En remplaçant y_{min} par y_{max} dans (1), la frontière basse de $\Re_T$ est caractérisée.

2.1.2 Estimation de la signature

De la phase précédente, une signature s_{CB} monodimensionnelle résumant au mieux la forme graphique du code-barres, doit être tirée. Pour ce faire, tous les pixels de chaque colonne de $\Re_T$ sont sommés dans le but de former un vecteur de longueur $x_{max} - x_{min} + 1$:

$$s_T(n) = \frac{1}{y_{max} - y_{min} + 1} \sum_{y=y_{min}}^{y_{max}} I(n + x_{min}, y) \text{ avec } n = [0, x_{max} - x_{min}]. \quad (2)$$

En traçant la variation des valeurs au sein du vecteur s_T, une première visualisation de la configuration du code-barres est obtenue. En effet comme le montre la figure 1, les valeurs les plus élevées font référence à une colonne de pixels de $\Re_T$, dans les tons de couleurs blanches, tandis que les valeurs les plus basses font référence à une colonne de pixels de $\Re_T$ dont la couleur est plus proche du noir.

En binarisant la signature préalablement recueillie, il est alors envisageable de supprimer les parties excédantes, dues à la manipulation de l'opérateur, sur les côtés gauche et droit de $\Re_T$. Pour matérialiser ces

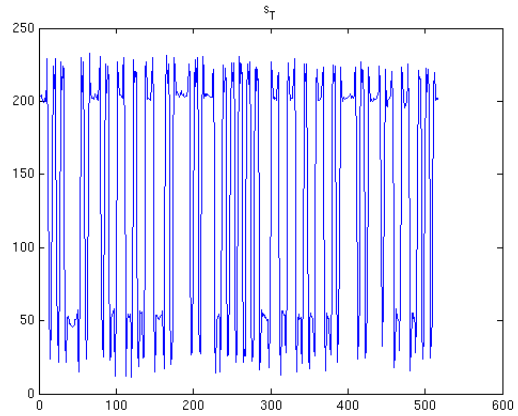

FIGURE 1 – Tracé de s_T pour le code-barres fourni dans le sujet

propos, il faut avant tout calculer un seuil pour distinguer la population qui se verra attribuer d'un 1 ou ou d'un 0. Ce dernier est évalué grâce à l'algorithme d'Otsu :

$$\text{crit}(k) = w(k)[\mu(N-1) - \mu(k)]^2 + (1 - w(k))\mu(k)^2 \text{ avec } h \text{ un histogramme de } \Re_T \text{ sur } N \text{ classes} \qquad (3)$$

$$w(k) = \frac{\sum_{i=0}^{k} h(i)}{\sum_{i=0}^{N-1} h(i)} \text{ et } \mu(k) = \frac{\sum_{i=0}^{k} ih(i)}{\sum_{i=0}^{N-1} h(i)}.$$

Le critère d'Otsu affiché (figure 2) possède une forme en cloche rectangulaire. Plus l'image contient une population de pixels bien distincte, plus la forme rectangulaire est accrue.

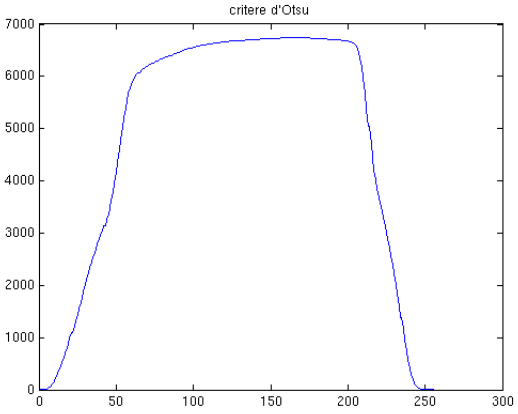

FIGURE 2 – Critère d'Otsu pour le code-barres fourni dans le sujet

Après que le seuil soit établi, il suffit de comparer chaque composante de s_T avec celui-ci. Si le seuil est inférieur au scalaire regardé, alors un 1 est attribué comme nouvelle valeur, sinon un 0. La signature devient donc un vecteur logique dans lequel il est plus simple d'identifier chaque barre du code (figure 3). De plus le début et la fin de ce nouveau vecteur ressemblent à quelque chose du type :

$$[\underbrace{1\cdots1}_{n \text{ fois}}0101\cdots1010\underbrace{1\cdots1}_{m \text{ fois}}]$$

.

Compte tenu du pourtour du code-barres, globalement blanc, le vecteur précédent possède une répétition de 1 en début et en fin du vecteur. Ensuite l'alternance de 1 et 0 annonce en fait les gardes normales des

codes-barres de la catégorie EAN 13. Elles établissent le commencement et la terminaison du code-barres, et offrent donc la possibilité d'enlever les parties superflues de l'ancien vecteur logique, pour se concentrer sur la signature réelle du code-barres s_{CB} et ainsi déchiffrer son contenu.

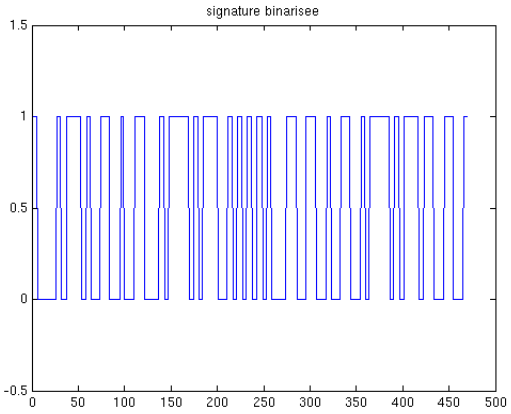

FIGURE 3 – Signature binarisée pour le code-barres donné dans le sujet

2.1.3 Identification des chiffres

La signature s_{CB} estimée auparavant est de longueur l et est segmentée en douze signatures partielles représentant chacune un numéro du code-barres. La découpe est opérée en fonction de l'estimation préalable de l'unité de base u, qui vérifie la relation $l = 89u$, car les gardes normales d'ouverture et de fermeture du code ont été retirées dans la section ci-dessus. Chaque signature partielle, s_p, ainsi jaugée est confrontée à l'ensemble des signatures théoriques permises afin d'identifier le chiffre candidat le plus probable. Or une signature théorique, s_{th}, ne contient que 7 bandes, il faut donc rééchantillonner cette dernière pour produire le résultat souhaité. Il s'agit de trouver les bornes des 7 intervalles d'entiers constituant chacun une des 7 bandes précédemment citées, afin que la concaténation des vecteurs formés par ces intervalles forme un vecteur de longueur M. L'application décrite ci-dessous accomplit la transformation attendue :

$$f : \begin{cases} [1,8] \to [1,M] \\ \\ x \mapsto \left\lfloor \frac{M-1}{7}(x-1) \right\rfloor + 1 \end{cases} \tag{4}$$

Cette fonction donne effectivement les bornes des différents intervalles des vecteurs évoqués ci-haut.

Dès que cette procédure est terminée, il ne reste plus qu'à calculer la corrélation entre s_p et s_{th} pour trouver la signature théorique la plus plausible :

$$c(s_p, s_{th}) = \frac{< s_{th} - \overline{s_{th}}, s_p - \overline{s_p} >}{\|s_{th} - \overline{s_{th}}\| \|s_p - \overline{s_p}\|} \tag{5}$$

Après que les douze signatures théoriques aient été recherchées, le premier chiffre du code barre-barres est décodé en opérant de la manière proscrite dans l'énoncer, et le calcul de la clé de contrôle pour valider le code dans son intégralité est acheminé.

Au terme de cette première partie, il incombe qu'un code-barres inscrit dans une image numérique dont les bords sont parallèles aux bandes du code, peut être déchiffré avec l'intervention, tout de même, d'un opérateur au départ de l'algorithme.

Néanmoins les images possédant une unité de base u inférieur à 2 ne peuvent être analysées adéquatement. Une raison possible à cette faille de l'algorithme vient du fait que, lorsque $u \in]1,2[$, alors une erreur sur le sur-échantillonnage devient fatale. En effet, les signatures partielles théoriques et expérimentales ne diffèrent que d'un bit si une erreur se produit, et par conséquent la signature partielle estimée change complètement.

2.2 Cas d'un code-barres orienté

Pour cette partie nous avons utilisé des images de code-barres orientés (type figure 4). L'opérateur doit toujours indiquer la largeur du code-barres via la procédure décrite dans la section précédente.

FIGURE 4 – Code-barres orienté

2.2.1 Détermination de l'angle de rotation

Dans notre cas la détermination de l'angle d'orientation est simplifiée. En effet, le code-barres dispose d'un nombre de lignes droites important, et l'angle qu'elles font avec les bords de l'image n'est autre que l'orientation du code-barres. L'utilisation de la transformée de Hough nous permet de détecter ces lignes. Pour cela nous devons au préalable détecter les contours de l'image. Ceci a été fait en utilisant le filtre de Sobel. Les lignes indiquant l'orientation du code sont dorénavant détectables par la transformée de Hough (figure 5).

Pour chaque pixel blanc, plusieurs lignes passant par ce même point sont tracées, et cela avec des angles différents. Pour chaque ligne précédemment définie, on détermine la distance ρ (c'est-à-dire la distance perpendiculaire à l'origine) ainsi que l'angle que forme la ligne avec l'axe des abscisses. Un graphique représentant les distances (ρ) des lignes pour chaque angle (θ), connu sous le nom d'espace de Hough, est alors créé (figure 6).

Chaque sinusoïde correspond aux lignes issues de chacun des pixels blancs considérés. Notre image est composées d'un réseau de lignes identiques dans leur orientation. Dans l'espace de Hough (figure 6) les points (lignes verticales à ϕ degrés) où les sinusoïdes se coupent donnent des distances et un même angle. Ces distances avec le même angle indiquent les lignes qui coupent les pixels testés, autrement dit le réseau de lignes parallèles du code-barres. Dans notre cas on remarque cette angle vaut 32 degrés.

2.2.2 Redressement de l'image

Une fois l'angle déterminé, il ne reste plus qu'à appliquer une rotation de ce même angle à l'image. Le décodage du code-barres reste ainsi le même que précédemment. La transformation étant une simple rotation, la taille/dimension du nouvel espace peut être facilement connu. En effet, il suffit de regarder la rotation des coins de l'image. Appliquons donc cette rotation aux coordonnées $(1,1),(w,1)(1,h)$ et (w,h) où w et h désignent respectivement la largeur et la hauteur de l'image considérée. Pour ce faire on utilise la matrice de

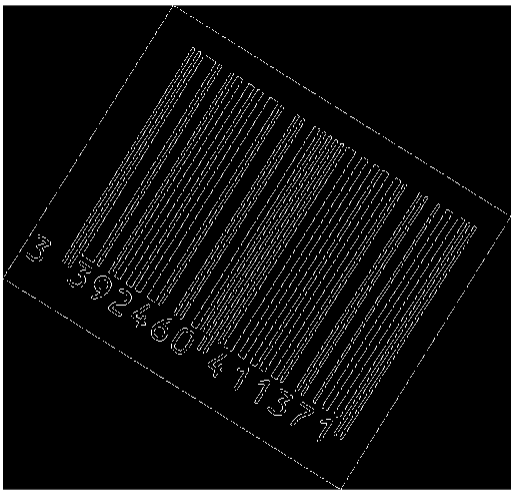

FIGURE 5 – Détection de contour d'un code-barres

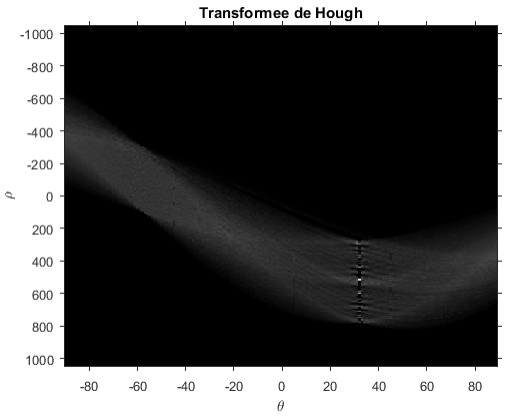

FIGURE 6 – Espace de Hough

rotation suivante :

$$R_\theta = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \text{ où } \theta \text{ est l'angle de rotation déterminé précédemment.} \quad (6)$$

Il ne nous reste plus qu'à prendre le minimum ainsi que le maximum, suivants les abscisses de ces nouvelles coordonnées puis suivant les ordonnées de ces coordonnées. Ainsi, l'espace d'arrivée est entièrement englobé ente x_{min} et x_{max} ainsi qu'entre y_{min} et y_{max}. Ces coordonnées peuvent être négatives mais cela ne pose pas de problème. À partir de là, il est aussi possible de connaître la taille de l'image redressée. En effet sa largeur n'est autre que $x_{max} - x_{min}$ et sa hauteur $y_{min} - y_{max}$. On peut désormais appliquer la rotation à l'ensemble de l'image.

À la suite des explications données ci-dessus, un algorithme pour faire pivoter l'image après que l'opérateur ait effectué la procédure d'initialisation a été implémenté. Cependant une deuxième limite a été rencontrée. Effectivement, les codes qui sont orientés dans le sens horaire présentent un défaut lorsqu'ils sont traités par la fonction créée. Les coordonnées renvoyées par les deux cliques de l'assistant ne subissent pas les mêmes effets de rotation que le reste de l'image. Ce qui a pour conséquence de ne pas pouvoir détecter correctement la région d'intérêt $_T$.

2.3 Cas d'un code-barre dégradé

Pour analyser la robustesse de notre algorithme nous avons utilisé des codes-barres dégradés à l'aide d'un logiciel de retouche tel que GIMP. Différents cas ont été analysés. Nous les commentons par la suite.

2.3.1 Dégradation de l'image en apposant une tâche colorée

Nous avons donc ajouté du noir entre les lignes du code, où la dominante de couleur à cette endroit est blanche (figure 7).

FIGURE 7 – Code-barres tâché

Nous avons pris ϵ égale à 0.2. Bien que la région d'intérêt ne soit pas totalement acquise, l'estimation de la signature reste inchangé et le décodage est juste.
Avec $\epsilon = 0.3$ la région d'intérêt est totalement acquise, et l'estimation de la signature reste juste même avec la prise en compte de toute la tâche lors de son estimation. Nous pouvons remarquer l'influence de la tâche sur la signature non binarisée en figure 8.

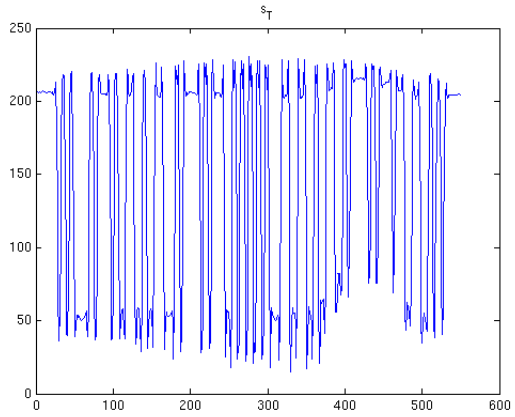

FIGURE 8 – Signature de l'image tâchée

La technique évaluant la région d'intérêt de la signature permet même à l'utilisateur spécifier un rectangle d'entrée incluant une tâche. Ces propos sont du moins avancés à la suite de l'expérience réalisée, car l'algorithme a estimé correctement la signature.

2.3.2 Niveau d'éclairage de l'image de départ modifié

Une dernière perturbation a été mise en place, pour de nouveau mettre à l'épreuve les performances des travaux accomplis. Une dégradation de la luminosité fût imputer à une image, qui est ensuite passée dans

la chaîne de traitements. Malheureusement, le déchiffrage du code-barres présent sur cette image, c'est mal passé. En effet, le seuil calculé par le critère d'Otsu (2) n'est pas cohérent avec la population de pixels observé. Cette nouvelle défaillance entraîne aussi une erreur lors de la binarisation de l'image et donc de la procédure de décodage.

3 Conclusion

Pour conclure, le projet mené lors de ces quelques semaines, ne s'est pas avéré totalement concluant. Bien que des fonctionnalités de bases permettant de décoder des codes-barres non orientés et orientés dans le sens trigonométrique ont été élaborées, l'algorithme proposé doit encore être amélioré pour pouvoir par exemple intégrer un module de caisse automatique dans un supermarché. Effectivement, comme il a été remarqué dans les différentes parties de ce rapport, l'ensemble des opérations conduisant au déchiffrement d'un code ne sont pas toutes parfaites.

Malgré ces désagréments, il fût intéressant de mettre au point cette chaîne de traitements, et de lui faire passer des tests d'efficacité. Ce dernier aspect est d'autant plus essentiel, car il doit être mené à chaque fois qu'un ingénieur propose une solution à un problème donné.

4 Bilan organisationnel

Pour réaliser au mieux ce projet, les tâches entreprises se sont d'abord concentrées sur le travail nécessaire à rendre i.e les étapes de décodage d'une image contenant un code-barres non orienté ont toutes été suivies ainsi que la phase de décodage d'un code-barres orienté.

La majorité du travail a été effectuée lors des séances consacrées à l'avancement du projet. Pendant ces cours nous essayions d'abord de trouver des solutions algorithmiques à l'aide de preuves graphiques et mathématiques ébauchées de manière manuscrite. Puis nous passions à une phase d'implémentation qui se basait plus sur un échange que sur une mobilisation du poste par l'un de nous deux. Cependant pour les étapes plus délicates telles que le sur-échantillonnage des signatures théoriques et la rotation de l'image, quelques heures de recherches supplémentaires ont dû être inévitables. En effet après la deuxième séance, une après-midi a été dédiée à la recherche d'une solution pour le sur-échantillonnage des signatures partielles théoriques. Au terme de ces travaux personnels, nous avions trouvé un premier algorithme qui ne fonctionnait qu'avec l'image fournie par le sujet.

Après la troisième séance, nous avons enfin mis au point un algorithme de sur-échantillonnage de signatures partielles qui fonctionnent avec l'ensemble des images de notre base de données. L'algorithme pour la rotation de l'image a quant à lui été établi en plusieurs parties. Tout d'abord lors de la quatrième séance, une première ébauche fût modelée, mais elle n'était pas optimale. En effet son temps d'exécution était assez conséquent et nous ne pouvions pas procéder à la phase d'initialisation effectuée par l'opérateur. Après la quatrième séance, le traitement des codes orientés est donc devenu notre principal source de travaux. Différents algorithme ont été mis au point, mais comme il a été mentionné dans la section consacrée, une solution optimale n'a pas été trouvée. Enfin pour tester la robustesse de nos algorithmes, des dégradations ont été appliquées à des images, et donc du temps supplémentaire dans la réalisation de ce projet a été ajouté.

5 Annexe

```
clc; clear ; close all;

%% Initialisation des variables
image = imread('image/barcode5.jpg');
image = double(image(:,:,1));
ep = 0.2;
N = 256;

% Traitement de l'image
% Recuperation des coordonnees de l'operateur ainsi que de l'angle de
% rotation de l'image
[xmin, xmax, ymin, ymax] = get_input(image);
angle = get_teta(image);

% Redressement de l'image
[new_image, min_x, max_x, min_y, max_y] = ...
my_imrotate(image, xmin, xmax, ymin, ymax, angle);

figure,
imshow(uint8(new_image));

figure,
test = new_image;
test(min_y:max_y, min_x:max_x) = 127;
imshow(uint8(test));
%%
% Suppression des zones superflues au dessus du code-barres
[ROI, min_x, max_x, min_y, max_y] = ...
get_ROI(new_image, ep, min_x, max_x, min_y, max_y);

% Projection dans l'espace a une dimension
s = get_signature(new_image, min_y, max_y, min_x, max_x);

% Binarisation de la signature prealablement calculee
[crit, ind, sb, xcenter] = binarize(N, ROI, s);

% Decodage du code-barres et verification que le procede s'est opere
% correctement
[final, classe] = estimate_signature(sb);
if (length(final) ~= 1)
    res = classe2nb(final, classe);
    if (res(1) == -1 || sum(res) == 0)
        disp('Barcode is wrong ! :(');
    else
        bool = control_key(res);
    end
end
```

```
function [new_image, min_x, max_x, min_y, max_y] = my_imrotate(image, xmin, xmax, ymin, ymax, angle)
% fonction realisant la rotation de l'image et des entrees faites par l'operateur

        rads = angle*pi/180;

        [h1, w1] = size(image);
        c = [0, 0];

        [x, y] = rotation([xmin, xmax], [ymin, ymax], -rads, c);
        xmin = floor(min(x));
        xmax = floor(max(x));
        ymin = floor(min(y));
        ymax = floor(max(y));

        [x, y] = rotation([1, w1, 1, w1], [1, 1, h1, h1], -rads, c);
        x_min = floor(min(x));
        x_max = floor(max(x));
        y_min = floor(min(y));
        y_max = floor(max(y));

        [Xtmp, Ytmp] = meshgrid(x_min:x_max, y_min:y_max);
        new_image = ones(y_max-y_min+1, x_max-x_min+1)*255;
        [h2, w2] = size(new_image);
        [X, Y] = rotation(Xtmp, Ytmp, rads, c);
        X = floor(X);
        Y = floor(Y);
```

```
27
28          for i=1:w2
29              for j=1:h2
30                  if (Y(j,i) > 0 && Y(
                        j,i) <= h1 &&  X(
                        j,i) > 0 && X(j,i
                        ) <= w1)
31                      new_image(j,i) =
                            image(Y(j,i)
                            , X(j,i));
32                  end
33              end
34          end
35
36          [~, min_x] = ismember(xmin,
                Xtmp');
37          [~, max_x] = ismember(xmax,
                Xtmp');
38          [~, min_y] = ismember(ymin,
                Ytmp);
39          [~, max_y] = ismember(ymax,
                Ytmp);
40
41  end
```

```
1   function [crit, ind, sb, xcenter] =
        binarize(N, ROI, signature)
2   % binarisation de la signature
3       [h, xcenter] = hist(ROI(:),N);
4
5       figure
6       stem(xcenter, h);
7       title('histogramme de la region
            d''interet');
8       xlabel('classe de pixels');
9
10      crit = 0;
11
12      for k = 1:N
13          w(k) = sum(h(1:k))/sum(h);
14          s = 0;
15          for i = 1:k
16              s = s + i*h(i);
17          end
18          mu(k) = s/sum(h);
19      end
20
21      for k = 1:N
22          crit(k) = w(k)*(mu(N)-mu(k))
                ^2 + (1-w(k))*mu(k)^2;
23      end
24
25      figure,
26      plot(crit);
27      title('critere d''Otsu');
28
29      [~, ind] = max(crit)
30
31      sb = signature > ind;
32
33      [beg_ind, end_ind] =
            crop_signature(sb);
34
35      sb = sb(beg_ind:end_ind);
36
37      figure,
38      plot(sb);
39      title('signature binarisee');
40      ylim([-0.5 1.5]);
41
42
43  end
```

```
1   function [est_signature] = classe2nb
        (est_signature, classe)
2   % fonction qui determine le premier
        nombre via la procedure decrite
        dans le
3   % sujet
4       el1 = [1 1 1 1 1 1; 1 1 2 1 2 2;
            1 1 2 2 1 2;...
5               1 1 2 2 2 1; 1 2 1 1 2
                    2];
6       el2 = [1 2 2 1 1 2; 1 2 2 2 1 1;
            1 2 1 2 1 2;...
7               1 2 1 2 2 1; 1 2 2 1 2
                    1];
8
9       el = [el1; el2];
10
11      for i = 1:length(el(:,1))
12          if (isequal(classe,el(i,:)))
13              res = i-1;
14              est_signature = [res
                    est_signature];
15          end
16      end
17
18      if (length(est_signature) == 12)
19          est_signature = [-1
                est_signature];
20      end
21
22  end
```

```
1   function [bool] = control_key(
        est_signature )
2   % fonction qui calcule la clef de
        controle et qui permet de valider
```

```
    le
% code-barres dechiffre

    last = 0;
    for i=1:2:length(est_signature)
        -1
        last = last + est_signature(
            i);
    end

    tmp = 0;
    for i=2:2:length(est_signature)
        -1

            tmp = tmp +
                est_signature(i);

    end
    last = last + 3*tmp;

    last = mod(last, 10);
    key = 10 - est_signature(13);
    key = mod(key,10);

    if (key == last)
        bool = 1;
        disp('Barcode is right ! :)')
            ;
    else
        bool = 0;
        disp('Barcode is wrong ! :('
            );
    end

end
```

```
function [est_nb, classe, u] =
    estimate_signature(sb)
% fonction permettant d'estimer la
    signature
% sb : signature binarisee (vecteur
    ligne)
% est_nb : le vecteur final
    contenant les nombres decodes

l = length(sb);
u = l/89;

if (u > 1 && u < 2)
    disp('Shannon theorem not
        checked !');
    est_nb = 0;
    classe = 0;
else
    el_th = table_el_th();
    classe = zeros(1,6);
    ya=1;
    xb=8;
    xa=1;

% on determine la taille d'une
    signature partielle dans la
    signature
% binarisee puis on sur-
    echantillonne toutes les
    signatures theoriques
% afin de pouvoir calculer la
    correlation entre la signature
% partielle et ces signatures

% pour les six premiers chiffres, on
     ne prend pas en compte la garde
% normale, il en est de meme pour
    les six derniers chiffres.

    for i = 1:6
        sig_ex = sb(floor((i-1)*7*u)
            +1:floor(i*7*u));
        sp = sig_ex - mean(sig_ex);
        den2 = norm(sp);
        yb = length(sig_ex);
        y = (yb-ya)/(xb-xa);
        for k = 1:length(el_th(:,1))
            sig_th = zeros(1,yb);
            for j = 1:7
                ind1 = floor(y*(j-1)
                    )+1;
                ind2 = floor(y*j)+1;
% on calcule l'intervalle de sur-
    echantillonnage pour
% la signteure theorique dont les
    bornes sont ind1 et
% ind2
                if (j ~= 1)
                    ind1 = ind1 + 1;
                end
                sig_th(ind1:ind2) =
                    el_th(k,j).*ones
                    (1,length(ind1:
                    ind2));
            end

            sth = sig_th - mean(
                sig_th);
            den1 = norm(sth);
% intercorrelation entre la
    signature partielle et la
    signature
```

```
% theorique
                c(k) = sum(sth.*sp)/(
                    den1*den2);

            end

            [~, ind] = max(c); %
                determination de la
                signature partielle
% par maximisation de la
    intercorrelation
            [est_nb(i), classe(i)] =
                ind2sym(ind);

        end

        for i = 7:12
            sig_ex = sb(floor(5*u+(i-1)*
                u*7)+1:floor(i*u*7+5*u));
            sp = sig_ex - mean(sig_ex);
            den2 = norm(sp);
            yb = length(sig_ex);
            y = (yb-ya)/(xb-xa);
            for k = 1:length(el_th(:,1))
                sig_th = zeros(1,yb);
                for j = 1:7
                    ind1 = floor(y*(j-1)
                        )+1;
                    ind2 = floor(y*j)+1;
                    if (j ~= 1)
                        ind1 = ind1 + 1;
                    end
                    sig_th(ind1:ind2) =
                        el_th(k,j).*ones
                        (1,length(ind1:
                        ind2));
                end

                sth = sig_th - mean(
                    sig_th);
                den1 = norm(sth);

                c(k) = sum(sth.*sp)/(
                    den1*den2);
            end

            [~, ind] = max(c);
            [est_nb(i), ~] = ind2sym(ind
                );

        end
    end

end
```

```
function [xmin, xmax, ymin, ymax] =
    get_input(image)
% fonction qui recupere les
    coordonnees de l'operateur
% on reaorganise les coordonnees (
    min, max) pour obtenir le
    rectangle
% selectione

imshow(uint8(image));
title('image de depart');

[x, y] = ginput(2);
x = floor(x);
y = floor(y);

xmin = min(x);
xmax = max(x);

ymin = min(y);
ymax = max(y);
end
```

```
function [ ROI, xmin, xmax, ymin,
    ymax ] = get_ROI( image, ep, xmin
    ,...

% fonction recuperant le rectangle
    determine par l'operateur
    data = double(image(ymin:ymax,
        xmin:xmax));
    figure,
    imshow(uint8(data));
    title('rectangle recupere par l'
        'operateur');

    reference = data(1,:);
    den = sum(reference);
    num = den;
    ratio = den/num;

    while(1-ep < ratio && ratio < 1+
        ep)
        ymin = ymin-1;
        if (ymin ~= 0)
```

```
            num = sum(double(image(ymin,xmin:xmax)));
            ratio = num/den;
        else
            ymin = 1;
            break;
        end
    end

    reference2 = data(end,:);
    den2 = sum(reference2);
    num2 = den2;
    ratio2 = den2/num2;


    while(1-ep < ratio2 && ratio2 < 1+ep)
        ymax = ymax+1;
        if (ymax ~= length(image(:,1)))
            num2 = sum(double(image(ymax,xmin:xmax)));
            ratio2 = num2/den2;
        else
            ymax = length(image(:,1));
            break;
        end
    end

    ROI = image(ymin:ymax,xmin:xmax);
    figure,
    imshow(uint8(ROI))
    title('region d''interet en hauteur');

end
```

```
function [s] = get_signature(image, ymin, ymax, xmin, xmax)
% fonction qui projette l'image 2D dans un vecteur
% cette fonction nous permet de recuperer de projeter la region d'interet
% dans un vecteur
    for n = 0:xmax - xmin
        s_tmp = 0;
        for y = ymin:ymax
            s_tmp = s_tmp + double(image(y,n+xmin));
        end
        s(n+1) = (1/(ymax+1-ymin))*s_tmp;
    end

    figure,
    plot(s);
    title('s_T');

end
```

```
function [angle] = get_teta(image_rot)
% fonction qui estime la rotation de l'image via la transformee de Hough

    % on detecte les contours par la methode sobel
    edge_image = edge(double(image_rot), 'sobel');

    [H, T, R] = hough(edge_image);

    % on recupere l'indice (angle) en cherchant la colonne
    % dont la variance est maximale
    [~, ind] = max(var(H,0,1));
    angle =  (ind-1) - 90;

    %Affichage
    figure, imshow(uint8(H),'XData', T,'YData',R,...
        'InitialMagnification','fit');
    title('Transformee de Hough');
    xlabel('\theta'), ylabel('\rho');
    axis on, axis normal, hold on;
end
```

```
function [ymin, ymax, xmin, xmax, ROI]=height_of_barcode(image, ep)
% fonction permettant de supprimer les zones superflues se trouvant au
% dessus et en dessous de la region d'interet

    imshow(uint8(image));
    image2=image;
    [x, y] = ginput(2);
    x = floor(x);
    y = floor(y);

    xmin = min(x);
    xmax = max(x);
```

```
13
14      ymin = min(y);
15      ymax = max(y);
16
17      data = image2(ymin:ymax,xmin:
            xmax);
18      data = double(data);
19      reference = data(1,:);
20
21      den = sum(reference);
22      num = den;
23      ratio = den/num;
24
25      while(1-ep < ratio && ratio < 1+
            ep)
26          ymin = ymin-1;
27          num = sum(double(image2(ymin
                ,xmin:xmax)));
28          ratio = num/den;
29      end
30
31      reference2 = data(end,:);
32      den2 = sum(reference2);
33      num2 = den2;
34      ratio2 = den2/num2;
35
36
37      while(1-ep < ratio2 && ratio2 <
            1+ep)
38          ymax = ymax+1;
39          num2 = sum(double(image2(
                ymax,xmin:xmax)));
40          ratio2 = num2/den2;
41      end
42
43      ROI = image2(ymin:ymax,xmin:xmax
            );
44      imshow(uint8(ROI))
45  end
```

```
1   function [nb, classe] = ind2sym(ind)
2   % fonction qui convertit un indice
        faisant reference a une des
        signatures
3   % partielles de la table de l'
        ensemble des signatures
        partielles, en la
4   % classe qui correspond a cette
        signature partielle
5
6       if (1 <= ind && ind <= 10)
7           classe = 1;
8       end
9
10      if (11 <= ind && ind <= 20)
11          classe = 2;
12      end
13      if (21 <= ind && ind <= 30)
14          classe = -1;
15      end
16
17      tmp = mod(ind,10);
18      if (tmp == 0)
19          nb = 9;
20      else
21          nb = tmp - 1;
22      end
23
24  end
```

```
1   function [X, Y] = rotation(X, Y,
        rads, C)
2   % fonction realisant la rotation de
        plusieurs point dont les
        coordonnees en
3   % abscisse de ces points sont
        specifiees dans le vecteur X, et
        les
4   % coordonnees en ordonne dans le
        vecteur Y
5
6       Xtmp = X - C(1);
7       Ytmp = Y - C(2);
8       X = Xtmp*cos(rads) - Ytmp*sin(
            rads) + C(1);
9       Y = Xtmp*sin(rads) + Ytmp*cos(
            rads) + C(2);
10
11  end
```

```
1   function [el]=table_el_th()
2   % fonction repertoriant l'ensemble
        des signatures partielles
        theoriques
3
4       elA = [1 1 1 0 0 1 0; 1 1 0 0 1
            1 0; 1 1 0 1 1 0 0;...
5               1 0 0 0 0 1 0; 1 0 1 1 1
                    0 0; 1 0 0 1 1 1 0;...
6               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];
7
8       elB = [1 0 1 1 0 0 0; 1 0 0 1 1
            0 0; 1 1 0 0 1 0 0;...
9               1 0 1 1 1 1 0; 1 1 0 0 0
                    1 0; 1 0 0 0 1 1 0;...
10              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];
11
12      elC = [0  0  0  1  1  0  1; 0 0 1 1 0
            0  1; 0 0 1 0 0 1  1;...
13              0 1 1 1 1 0 1; 0 1 0 0 0
                    1 1; 0 1 1 0 0 0 1;...
14              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];
15
16      el = [elA; elB; elC];
17
18  end
```