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 **5**

5 Conclusion **5**

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.

  language=Matlab [frame=single] for y=y$_m in : -1 : 1 r = abs(sum(img_n g(y, x_m in : x_m ax))/r_m in); if r > 1 + epsilon || r < 1 - epsilon y_m in = y; break; end$

- **Identification des chiffres**

4 Résultats

5 Conclusion