Projet : Image et Template Matching: verifier le nombre de timbres d'une planche de timbres

Notions : Allocation dynamique, fichier, structure,travail en groupe

L'objectif de ce projet est de vérifier le nombre de timbres présents sur une planche de plusieurs timbres à l'aide de l'image parfaite d'un timbre. On dispose donc au depart d'une image de la planche de timbres et d'une image d'un seul timbre (figure 1). Pour retrouver la position et le nombre de timbres, nous allons utiliser la notion de Template Matching.

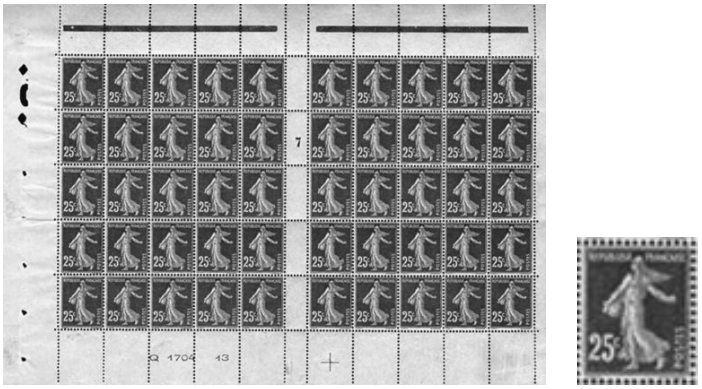

FIGURE 1 – Une planche et modèle de timbre

1 Quelques éléments sur l'image

Une image est une fonction $I(x, y)$ à support fini (ses dimensions) et dont les valeurs sont ici des scalaires sur 8 bits : le noir correspond à une valeur nulle, le blanc à 255.

En informatique, c'est simplement un tableau d'octets non signés (`unsigned char`) à 2 dimensions.

Vous avez figure 2 deux tableaux représentant les valeurs de l'image du tangram sur un voisinage de 6x6 pixels dans le fond et sur un sommet de triangle. Les valeurs de l'intensité ne sont pas uniformes et comportent un aléa dû au bruit d'acquisition.

FIGURE 2 – Images, octets et valeurs numériques

2 Template matching

Le Template matching consiste à calculer la ressemblance entre un modèle sous forme d'une petite image et une image pour chaque pixel de l'image. On calculer donc la distance entre l'image d'un modèle de dimension `nlm,ncm` et la portion de l'image centrée en `i,j`, comprise dans le rectangle de coordonnées `i-nlm/2,j-ncm/2`, `i+nlm/2,j+ncm/2`. La distance entre le modèle T et la partie de l'image I centrée en `i,j` est simplement l'écart quadratique moyen (EQM) :

$$d^2_{T,I}(i,j) = \frac{1}{nlm.ncm}. \sum_{di=0}^{nlm-1} \sum_{dj=0}^{ncm-1} \left(I[i - nlm/2 + di][j - ncm/2 + dj] - T[di][dj]\right)^2$$

Cette distance est nulle lorsque l'image `I` en `i,j` correspond exactement au modèle `T`. Sur des données réelles, le bruit est un phénomène aléatoire qui rend cette hypothèse peu vraisemblable. L'acquisition d'une image d'un même objet à des instants différents ne donne jamais des images rigoureusement identiques. On ne peut donc pas se contenter de rechercher les endroits où cette distance est nulle pour déterminer la présence du modèle.

Une heuristique (hypothèse souvent vérifiée mais parfois fausse) commune se base sur l'observation suivante : la distance $d_{T,I}(i,j)$ présente souvent un minimum local au point de coordonnées `i,j`.

Un minimum local est un point de coordonnées `i,j` dont la valeur est inférieure strictement à celle de tous ses voisins qui sont les points de coordonnées (i-1,j), `(i+1,j)`, `(i-1,j-1)`, `(i,j-1)`, `(i+1,j-1)`, `(i-1,j+1)`, `(i,j+1)`, `(i+1,j+1)`. La figure 3 illustre cette définition : les axes x et y sont ceux du support spatial de l'image, l'axe z est la valeur de la distance entre le modèle et une partie de l'image de la figure 1.

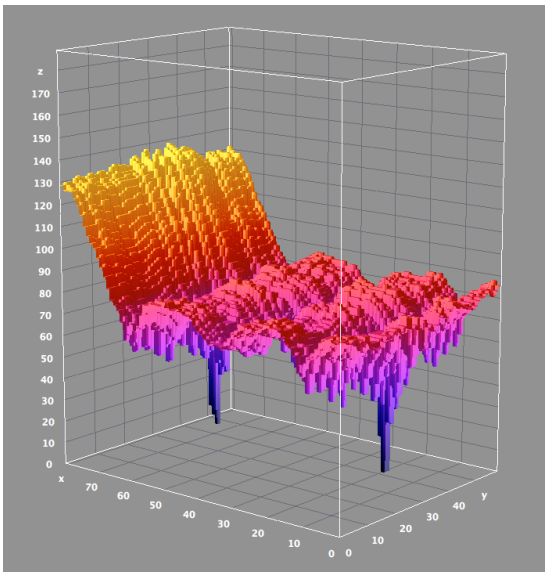

FIGURE 3 – Distance entre le modèle et une partie de la planche de la figure 1

On remarque 2 propriétés :

1. deux positions sont des minima locaux (bleu foncé) et correspondent effectivement à des timbres.
2. de nombreux autres minima locaux existent mais ont une valeur de distance beaucoup plus forte.

Pour déterminer la présence ou non du modèle et sélectionner uniquement les valeurs les plus faibles, l'heuristique sera donc d'avoir une distance minimale localement (propriété 1 ci dessus) **et** d'avoir une distance faible (propriété 2 ci dessus). On ne conservera que les minima locaux dont la valeur est inférieure à un seuil (qui dépend de l'image testée). Cette dernière opération se nomme seuillage.

En résumé, la méthode pour trouver le nombre de modèles T de taille nlm, ncm présents dans l'image I de taille nli, nci est donc la suivante :

1. Calculer la distance $d^2_{T,I}(i,j)$ pour $(i,j) \in [0..nli-1] \times [0..nci-1]$. Cette distance sera stockée dans une matrice d'entiers (et non d'octets) `imdistance` dont la structure est similaire à une image.
2. Déterminer les minima locaux de `imdistance` ; Dans une nouvelle matrice `im2` d'`unsigned char` (i.e. une image), mettre les valeurs des minima locaux à la valeur $\sqrt{d^2_{T,I}}$. Mettre toutes les valeurs des autres points à `UCHAR_MAX`. La constante `UCHAR_MAX` est définie dans le fichier d'entête `limits.h`.
3. Seuiller les points positifs de `im2` i.e. conserver les points de `im2` dont la valeur est comprise entre 0 et un seuil choisi par l'utilisateur (la valeur par default de nos exemples est 40).
4. Compter le nombre de valeurs strictement positives restantes de `im2` : c'est le nombre de modèles présents

3 Travail à réaliser

Ce projet se réalise en binôme et demande un bonne coordination entre les 2 membres du binôme. Il se réalise avec beaucoup de travail en dehors de la séance encadrée.

3.1 Méthode

La première étape consiste à analyser le problème et à le décomposer en sous problèmes plus simples. Pour donner un exemple, notre problème se décompose en 6 étapes :

1. lire l'image à partir d'un fichier
2. lire le modèle à partir d'un fichier
3. Calculer la matrice des distances entre l'image et le modèle
4. Déterminer les minima
5. Seuiller les minima
6. Afficher le nombre de minima

Attention : Le rôle de ces différentes étapes n'est pas suffisamment précis et il vous faut le compléter afin de pouvoir écrire le code, en particulier bien préciser les paramètres, leur rôle et leur mode de passage.

En itérant cette démarche d'analyse descendante sur chaque étape, on décompose le problème en plusieurs fonctions simples à coder (5 à 10 lignes de code C). Compiler et tester les fonctions les unes après les autres (développement incrémental). Vous pouvez utiliser les fonctions de lecture et écriture d'image que vous avez écrites lors du tp sur les fichiers et les structures. Vous devez vous partager le travail au sein de votre binôme.

3.2 Types et fonctions à écrire

Les types et les fonctions suivantes peuvent vous être utiles à écrire.

```
typedef unsigned char  PIXEL;
// Definition du type IMAGEUCHAR pour des images 8 bits d'octets
typedef struct {
  int nl,nc;
  PIXEL** val;
} IMAGEUCHAR;
// Definition du type IMAGEUCHAR pour des images 32 bits d'entiers
typedef struct {
  int nl,nc;
  int** val;
} IMAGEINT;
// Allocation de l'espace memoire pour les pixels
PIXEL** allouеMemoireImageChar(int nl, int nc);
// Creation  d'une image de d'unsigned char avec allocation dynamique de l'espace des pixels
IMAGEUCHAR creationImageUChar(int nl, int nc) ;
// Reset l'image avec liberation de l'espace memoire des pixels, mise a 0 des nombres de lignes et de
   colonnes
void resetImageUChar(IMAGEUCHAR* pim);
// Creation avec allocation d'une image de Int avec allocation dynamique de l'espace des entiers
IMAGEINT creationImageInt(int nl, int nc) ;
// Reset l'image avec liberation de l'espace memoire des entiers, mise a 0 des nombres de lignes et de
   colonnes
void resetImageInt(IMAGEINT* pim);
 // Lecture d'une image dans un fichier
int lectureImagePgmBinaire(char* fic,IMAGEUCHAR * im) ;
// Distance d^2 entre le modele et l'image au pixel de coordonees is, js
int distance( IMAGEUCHAR im, IMAGEUCHAR patch, int is,int js);
 // Calcul de la distance au modele pour tous les points de l'image.
IMAGEINT templateMatching(IMAGEUCHAR im, IMAGEUCHAR modele) ;
// Determination des minima locaux
IMAGEUCHAR minLocal(IMAGEINT im) ;
// Seuillage des valeurs positives inferieures a un seuil
IMAGEUCHAR seuilImageUChar(IMAGEUCHAR im, int seuil) ;
// Compter le nombre de valeurs non nulles
int valImageNonNulle(IMAGEUCHAR im) ;
// Si besoin, Ecriture d'une image dans un fichier
int ecritureImagePgmBinaire(char* fic, IMAGEUCHAR im) ;
```

3.3 Remarques d'implémentation

— Commencer par tester vos programmes sur de petites images (modele2.pgm et timbres10.pgm par exemple)

— Dans les fonctions que vous devez écrire, et particulièrement dans la fonction `distance`, vous pouvez avoir avoir besoin de pixels qui n'existent pas et sont en dehors de l'image. C'est en particulier le cas lorsque on calcule la distance entre le modèle et le coin supérieur gauche de l'image, de coordonnées (0,0). Pour simplifier l'écriture des fonctions, on utilisera un prolongement par périodisation : l'image s'enroule sur elle-même, ce qui consiste à répliquer virtuellement l'image dans toutes les directions (droite, gauche, bas, haut, diagonales. Si on cherche à accéder à un pixel de la ligne -1, on accédera au pixel de la dernière ligne de l'image. Au lieu d'utiliser l'accès par $im.val[i][j]$, on utilisera le modulo sous la notation $im.val[(i+im.nl)\%im.nl][(j+im.nc)\%im.nc]$

3.4 Données et programme de test

Les fichiers de modèles et d'images sont donnés sur le site des Tp et présents sur les machines de l'école dans le répertoire "/users/prog1a/C/librairie/projetS12019".

Une version binaire de ce projet est disponible dans ce répertoire sous le nom `projetS1`. Elle s'utilise avec 3 paramètres dont le dernier est optionnel sous la forme : `./projetS1 fichier_image fichier_modele seuil`. Le seuil est optionnel et fixé par default à 40.

Par exemple, `./projetS1 timbres/timbres15.pgm modeles/modele2.pgm 40` trouve 50 timbres de type `modele2.pgm` dans la planche de timbres `timbres15.pgm`

Le programme binaire fourni crée 2 fichiers utiles pour comprendre et debugger :

— un fichier image pgm de nom `matching.pgm` contenant le résultat du "template matching". Dans ce fichier, la distance $d^2_{T,I}(i,j)$ a été ramenée à une valeur comprise entre 0 et 255 pour pouvoir être visualisée. La valeur d'un entier x (32 bits) est ramenée sur un octet i (8 bits) par la formule : $i = x * 255/max(Image)$

— un fichier de nom `resultat.pgm` contenant l'image initiale sur laquelle on a superposé des croix aux endroits détectés

Remarque

Si vous souhaitez que votre programme puisse prendre en paramètre dans le Terminal les noms des fichiers à traiter, ou la valeur du seuil comme dans le code binaire fourni, il faut utiliser le prototype complet de la fonction main() en C. Voir ici : http://tdinfo.phelma.grenoble-inp.fr/2Aproj/fiches/prototype_main.pdf