Réalisation d'un panorama

Ludovic Lamarche
Quentin Perales
Elie Poussou

E.I.S.T.I
PAU

22 novembre 2013

Table des matières

1 Les six fonctions demandées dans le livrable **3**
1.1 Grayscale 3
1.2 Binnary 4
1.3 Histogramme 4
1.4 Dilate 5
1.5 Erode 6
1.6 Convolution [6] 7

2 Comment automatiser un panorama ? **9**
2.1 La détection des points d'intérêts d'une image 9
2.1.1 La transformée de Hough [4] [2] 10
2.1.2 Le detecteur de Harris 11
2.2 Relier les points d'intérêts : la méthode RANSAC [3] [7] . . . 11
2.3 Comparer les distances entre les points 12

Introduction

La seconde partie du projet nous a permis de réaliser les six fonctions demandées puis d'approfondir nos recherches pour automatiser la réalisation d'un panorama.
Afin d'expliciter notre travail, nous allons expliquer notre code pas à pas dans une première partie. Chaque fonction demandée a requis divers recherches. On pourra alors utiliser certains schémas pour expliquer les algorithmes utilisés dans notre programme.
Puis nous allons expliquer les différentes méthodes que nous pouvons choisir pour écrire le code qui premettra l'automatisation d'un panorama. On va pouvoir comprendre comment détecter les points d'intérêts, les relier puis enfin comparer les deux images pour les assembler.

Chapitre 1

Les six fonctions demandées dans le livrable

1.1 Grayscale

La fonction Grayscale permet de transformer une image couleurs en teintes de gris.
Dans une image en couleur, chaque pixel est codé par 3 composantes (Rouge, Vert et Bleu).
En grayscale, chaque pixel sera codé par une seule composante.
La formule utilisée est la suivante :
$pixelGrayscale = 0,299 * pixelRouge + 0,587 * pixelvert + 0,114 pixelbleu$
On parcourt donc l'image en couleurs, et on applique cette formule à chaque pixel, c'est à dire à ces trois composantes.
Cette fonction effectue donc une moyenne pondérée des pixels rouges, verts et bleus.
L'image obtenue sera donc 3 fois plus petite, et en noir et blanc.

1.2 Binnary

Chaque pixel d'une image binaire ne peut avoir pour valeurs que 0 ou 1. La fonction binary nécessite un seuil et une image en teintes de gris, le seuil étant choisit par l'utilisateur.

On parcourt l'image en teintes de gris, et pour chaque pixel, si son instensité est supérieure au seuil, on lui alloue la valeur 0 qui correspond à la couleur blanche. Si l'intensité du pixel est supérieure au seuil, on lui alloue la valeur 1 qui correspond à la couleur noire.
Voici l'algorithme utilisé :

```
POUR TOUT les pixels de l'image
    SI le pixel >seuil
    pixel=1
  FINSI
FINPOUR
```

1.3 Histogramme

L'histogramme d'une image permet d'illustrer la répartition des teintes d'une image.
Le but est de compter pour chaque intensité le nombre de pixels qui sont de cette intensité.

D'abord on crée un tableau de taille égale à la teinte maximale de l'image dont on veut obtenir l'histogramme.
Ensuite, on parcourt l'image et pour chaque pixel, on rajoute 1 dans la case du tableau qui correspond à l'instensité de ce pixel.

On peut résumer tout ceci grâce à l'algorithme suivant :

```
tab : tableau d'entiers de taille égale à la teinte maximale de
    l'image

POUR chaque pixel de l'image
tab[teinte_pixel]++
FINPOUR
```

On peut choisir de tracer cet histogramme mais le tableau suffit.

1.4 Dilate

Pour la dilatation nous avons utilisé un algorithme qui utilise des "object pixel" tiré du livre Machine Vision[5]. Un objet pixel est un groupe de neuf pixels formant un carré de taille 3x3 et qui respecte certaines conditions. Pour simplifier l'algorithme on nomme chaque case de 0 à 8.

3	2	1
4	8	0
5	6	7

Si on utilise les conventions de logique combinatoire, c'est à dire que le signe . signifie ET et le signe + signifie OU, les conditions d'un objet pixel se résume comme ceci :

$$
\begin{aligned}
&8.[((1+2+3).5+6+7).\overline{4}.\overline{0}] \\
&+[(1+0+7).(3+4+5).\overline{2}.\overline{6}] \\
&+[3.(5+6+7+0+1).\overline{2}.\overline{4}] \\
&+[1.(3+4+5+6+7).\overline{2}.\overline{0}] \\
&+[7.(1+2+3+4+5).\overline{0}.\overline{6}] \\
&+[5.(7+^{\circ}+1+2+3).\overline{4}.\overline{6}]
\end{aligned}
$$

Finallement, il suffit d'appliquer l'algorithme suivant :

```
POUR TOUT les pixels de l'image
  SI le pixel est un objet pixel ALORS
    copier le groupe de pixels dans l'image de destination
  FINSI
FINPOUR
```

Étant donné que cette méthode enlève beaucoup de pixels, il n'est pas nécessaire de faire une érosion en suivant. Cependant nous avons utilisé un autre algorithme moins restrictif pour l'erosion.

1.5 Erode

Pour l'érosion nous utilisons un masque. Nous testons un groupe de pixel et nous assignons la valeur 1 si tout les pixels dans le masque sont à 1. Sinon on retourne un 0. Nous avons pris directement l'algorithme d'opération pixel[4] :

```
destination : matrice de l'image érodée
source : matrice de l'image à éroder
x,y,k1,k2 : variables d'incrémentation

POUR CHAQUE pixel de l'image FAIRE //x et y étant la position
  POUR k1 allant de -1 à 1 FAIRE
    POUR k2 allant de -1 à 1 FAIRE
destination[x][y] = destination[x][y] ET source[x+k1][y+k2]
    FINPOUR
  FINPOUR
FIN POUR
```

1.6 Convolution [6]

La convolution permet d'appliquer un filtre de convolution à une image en échelle de gris. Le but est d'affecter pour chaque pixel de l'image des coefficients au pixels autour de celui-ci.

La taille du filtre est de taille variable, cependant, elle est toujours paire. Les filtres les plus utilisés sont les filtres de taille 3x3, mais parfois, les filtres de tailles supérieures sont requis. Les filtres sont récupérés en début de fonction, une erreur est générées si le filtre n'est pas correct, c'est à dire si le fichier texte comporte caractères autres que des chiffres ou nombres ou s'il n'y a pas assez de coefficients.

Afin d'expliquer l'algorithme qui mène à la convolution, nous allons utiliser un filtre 3x3 sur une image. Prenons le filtre F suivant :

$$\begin{matrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{matrix}$$

F est appliqué à un pixel p de coordonnées (x, y). La valeur du pixel après la convolution sera alors

$$nouvelleValeur = 0 * p(x-1, y-1) + 1 * p(x, y-1) + 0 * p(x+1, y-1)$$
$$+ 1 * p(x-1, y) + 1 * p(x, y) + 1 * p(x+1, y)$$
$$+ 0 * p(x-1, y+1) + 1 * p(x, y+1) + 0 * p(x+1, y+1)$$

De plus, on crée un décalage pour éviter d'assigner des coefficients à des pixels hors image. Pour un filtre 3x3, le décalage est de 1 ; pour 5x5, il est de 3 ; pour 7x7, il est de 5. On peut donc dire que

$$decalage = \frac{taille - 1}{2} \tag{1.1}$$

Voici un exemple de convolution que l'on peut obtenir :

$$\begin{matrix} 35 & 40 & 41 & 45 & 50 \\ 40 & 40 & 42 & 46 & 52 \\ 42 & 46 & 50 & 55 & 55 \\ 48 & 52 & 56 & 58 & 60 \\ 56 & 60 & 65 & 70 & 75 \end{matrix} \quad \text{X} \quad \begin{matrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{matrix} \quad = \quad \begin{matrix} 35 & 40 & 41 & 45 & 50 \\ 35 & 40 & 41 & 45 & 50 \\ 40 & 40 & 42 & 46 & 52 \\ 42 & 46 & 50 & 55 & 55 \\ 56 & 60 & 65 & 70 & 75 \end{matrix}$$

Le but va être maintenant d'utiliser ces fonctions ainsi que d'autres algorithmes pour automatiser la création d'un panorama.

Chapitre 2

Comment automatiser un panorama ?

Après avoir vu les principales fonctions utiles à notre panorama, nous devons maintenant développer des méthodes pour pouvoir coder des algorithmes capables d'assembler des images les unes aux autres et ainsi retrouver l'image initiale.

2.1 La détection des points d'intérêts d'une image

Dans un premier temps, il faut détecter les points d'intérêts de l'image, c'est-à-dire les points particulier de l'image, qui sont notables. Pour cela, nous allons comparer deux techniques : la transformée de Hough et le détecteur de Harris.

2.1.1 La transformée de Hough [4] [2]

Dans cette méthode, on utilise des images en noir et blanc, il faut donc convertir l'image couleur en une image en échelle de gris, puis la convertir en noir et blanc. Afin d'obtenir un nombre de points suffisamment important pour pourvoir les comparer mais pas trop pour éviter d'utiliser des points inutiles, on peut éroder ou dilater l'image.

Le but ici est de comparer les points les uns par rapport aux autres pour délimiter des figures géométriques précises. Le plus simple est de repérer les droites de l'image. On considère qu'une infinité de points passent par un point, la seule chose qui diffère est donc l'inclinaison de la droite qui passe par ce point. On calcule alors la norme algébrique ro du segment entre l'origine et la droite où le segment est perpendiculaire à cette droite ainsi que téta qui est l'angle d'inclinaison de la droite par rapport à la perpendiculaire à l'axe des ascisses passant par le point étudié.

On trace ensuite dans l'espace de Hough tous les couples ro (ordonnées) et teta (abscisses) obtenus pour chaque point. Les courbes de l'espace de hough sont concourrantes en plusieurs points qui sont alors les couples (ro, teta) caractéristiques de l'image.

Maintenant, il faut retracer les droites de Hough qui correspondent à l'image. Pour cela, il faut, pour chaque pixel, calculer un ro' et un téta', si ce couple est égal à un couple calculé précédement, lors on trace la droite qui correspond à ce couple.

On obtient ainsi toutes les droites caractéristiques de l'image.

2.1.2 Le detecteur de Harris

Nous avons utilisé l'algorithme de Harris trouvé sur operationPixel[4] et sur le blog de Crsouza[1].

```
A, B, C : Trois matrices pour le calcule de Harris
result : La liste des points de Harris
image : matrice de l'image source
threshole : variable qui reste à déterminer
DEBUT PROGRAMME:
A = deriveeVerticale(image);
B = deriveeHorizontale(image);
C = derivee(image);
POUR CHAQUE pixel de l'image FAIRE \\ x et y étant les
    coordonnées
  M = (A[y][x] * B[y][x] - C[y][x] * C[y][x])
  - (0.04 * ((A[y][x] + B[y][x]) * (A[y][x] + B[y][x])));
    SI (M>threshole)
      result = ajoutCoordonnee(result, x, y, M);
FINPOUR
```

Pour utiliser cet algorithme nous devons trouver la variable threshole qui varie selon les images à traiter. Les points importants d'une image détectés par Harris sont visualisables dans une image en noir et blanc crée en utilisant l'option " –harris ".

2.2 Relier les points d'intérêts : la méthode RANSAC [3] [7]

La méthode RANdom SAmple Consensus (abrégée RANSAC) est basée sur un modèle mathématique qui utilise les probabilités pour optimiser une trajectoire entre des points. Elle a été crée afin de réduire le temps de calcul pour des méthodes de "vote" classique comme la transformée de Hough.

On suppose que les points sont un ensemble d'inlers et d'outliers, les inliers sont les points qui vont correpondre à un modèle et les outliers qui eux ne vont pas correspondre à ce modèle.

RANSAC utilise dans un premier temps un ensemble de points choisis de manière aléatoire, puis on estime un modèle à partir de ce tirage. Ensuite, les autres données sont alors testées sur le modèle précédents et sont ajoutées aux inliers s'ils correspondent au modèle. Enfin, on considère qu'un modèle est juste si le nombre d'inliers est assez important. Le modèle est alors recalculé à partir des inliers, on calcule alors l'erreur de ce modèle. Ainsi, on associe une erreur au modèle de départ. Les autres modèles seront alors évalués par rapport au modèle précédent. Celui qui a l'erreur la plus faible est le modèle de référence pour l'image.

2.3 Comparer les distances entre les points

Nous avons essayé dans un cas parfait où l'image à été découpé par informatique de trouver les points en commun sur une image par rapport à une autre. Les distances et les couleurs sont donc exactement les mêmes. Nous avons utiliser l'algorithme de Harris pour selectionner les points clés sur chaque image et nous les avons comparé suivant cet algorithme :

```
1  Pour chaque points clé de l'image1
     Pour chaque points clé suivant de l'image1
3      Pour chaque points clé de l'image 2 égaux à l'image 1
         Pour chaque points clé suivant de l'image 2
5          Si la distance entre le point les deux points de l'image
       1 est égale à la distance des deux points de l'image 2 alors
             C'est un point en commun.
7          FinSi
         FinPour
9      FinPour
     FinPour
11 FinPour
```

Conclusion

Pour conclure, il nous reste maintenant la partie la plus compliquée du projet, à savoir automatiser la réalisation d'un panorama. Pour cela, nous allons mettre en oeuvre les fonctions déjà codées pour réaliser les différents algorithmes présentés dans la seconde partie. Tous seront tester pour voir lequel est le plus adapté en fonction des images en paramètre.

Bibliographie

[1] CESARSOUZA. *Harris Corners Detector in C#*. 2010. URL : http://crsouza.blogspot.fr/2010/05/harris-corners-detector-in-c.html.

[2] DEVELOPPEZ.COM. *[image] Détecteur de Ligne (Hough)*. 2008. URL : http://www.developpez.net/forums/d495285/autres-langages/algorithmes/contribuez/image-detecteur-ligne-hough/.

[3] Ezio MALIS et ERIC MARCHAND. *Méthode robustes d'estimation pour la vision robotique*. 2005. URL : http://www.irisa.fr/lagadic/pdf/2005_jnrr_malis.pdf.

[4] OPERATIONPIXEL@FREE.FR. *Traitement d'images*. 2010. URL : http://operationpixel.free.fr/pointinteret_fast.php.

[5] David VERSNON. *Machine Vision*. 1991. Chap. 4. URL : http://homepages.inf.ed.ac.uk/rbf/BOOKS/VERNON/Chap004.pdf.

[6] WIKIPÉDIA. *Kernel (Image processing)*. URL : http://en.wikipedia.org/wiki/Kernel_(image_processing).

[7] WIKIPÉDIA. *RANSAC*. URL : http://fr.wikipedia.org/wiki/RANSAC.