TP6: Optimisation au sens des moindres carrés

Le but de ce TP est d'implémenter une technique de recalage d'images qui utilise une méthode vue en cours d'analyse numérique : la minimisation au sens des moindres carrés.

Un recalage d'images consiste à mettre spatialement en correspondance deux images, afin de pouvoir les comparer ou combiner leurs informations respectives. Cette technique est utilisée pour de nombreuses applications comme le traitement de vidéos (pour le suivi de mouvement ou la compression), ou l'imagerie médicale (afin de fusionner plusieurs modalités d'imagerie).

Il existe dans la littérature un nombre considérable d'approches permettant d'estimer le mouvement entre deux images. Un algorithme de recalage d'images, qu'il s'inscrive dans le cadre d'une application interactive ou automatique, nécessite la définition de quatre critères:

1. **Les structures homologues.** Ce sont des informations extraites des images sur lesquelles va s'appuyer l'algorithme pour estimer le mouvement. Dans le cadre de ce TP, le mouvement est une variable cachée qui sera accessible par l'analyse des variations spatio-temporelles des niveaux de gris entre une image de référence et une image à recaler.

2. **La transformation spatiale à rechercher.** Dans le cadre de ce TP, nous nous limiterons à l'estimation d'une transformation à deux paramètres: une translation horizontale (notée ΔX) et une translation verticale (notée ΔY).

3. **Un critère caractérisant la qualité du recalage.** Dans le cadre de ce TP, le principe sera de calculer les paramètres de la transformation spatiale définie en 2 en minimisant la différence quadratique entre l'image de référence et l'image à recaler.

4. **Une stratégie de résolution** permettant de trouver la transformation optimisant le critère défini en 3. Nous exprimerons les paramètres de la transformation spatiale comme la solution d'un système surdéterminé que nous résoudrons à l'aide d'une minimisation au sens des moindres carrés.

Tout d'abord, la première section de ce TP se consacrera à quelques manipulations préliminaires d'images destinées à se familiariser avec les outils de base nécessaires pour la suite. Ensuite, la deuxième section visera à développer des routines permettant d'appliquer des transformations spatiales sur une image. Enfin, la troisième et dernière section de ce TP s'intéressera à estimer les paramètres d'une transformation spatiale simulée entre deux images.

I Manipulations d'images

Une image est modélisée par une matrice A, dont les valeurs sont des nombres réels compris entre 0 et 1, 0 correspondant au noir, 1 correspond au blanc, les autres valeurs correspondant à différents niveaux de gris. Nous allons d'abord nous familiariser avec la manipulation des images stockées au format PGM.

Téléchargez les deux fichiers suivants:

- **tp6.f90** : Définition des traitements à appliquer à l'image, c'est le fichier que vous devrez modifier.
- **image_reference.pgm** : Image en niveaux de gris sur laquelle vous travaillerez.

I.1 Négatif d'une image

Créez une routine qui prend en entrée une matrice A représentant l'image en noir et blanc et qui la modifie en son négatif. Vous appellerez cette routine dans le programme principal pour la tester. Vous sauvegarderez l'image modifiée à l'aide de la routine **save_pgm** afin de visualiser le résultat obtenu avec le logiciel de visualisation d'images **The Gimp** (qui se lance en tapant dans un terminal la commande "gimp").

I.2 Luminosité

Pour modifier la luminosité d'une image, il suffit de décaler toutes les valeurs des pixels soit vers le haut (éclaircissement), soit vers le bas (assombrissement). On note A_{ij} la valeur d'un pixel, sa nouvelle valeur sera A_{ij}^l où l représente la luminosité. Si $l = 1$, on ne change rien à l'image. Si $l < 1$, on éclaircit l'image et si $l > 1$, on l'assombrit. Créez une routine implémentant un tel filtre. Sauvegardez l'image au format PGM et visualisez le résultat obtenu avec **The Gimp**.

I.3 Détection de contours

Un filtre de détection de contours très simple est basé sur une approximation du Gradient. On considère que chaque pixel est la valeur en un point d'une fonction 2D discrète. Un contour est détecté lorsque le gradient a une valeur élevée. On crée alors une nouvelle image dont chaque pixel contient la valeur du gradient en ce point donné. On peut prendre, par exemple, juste la composante en x du gradient:

$$\partial_x A_{ij} = A_{i+1,j} - A_{i,j}$$

Codez cette détection de contour et appliquez la sur **lena.pgm**. Variez les plaisirs en prenant plutôt une approximation de $\|\nabla A_{i,j}\|$

Enfin, on peut utiliser l'approximation d'un Laplacien (qui ressemble à ce qu'on a utilisé dans l'approximation du Laplacien pour résoudre l'équation de Poisson):

$$\Delta A_{ij} = A_{i+1,j} + A_{i-1,j} + A_{i,j+1} + A_{i,j-1} - 4A_{i,j}$$

Comparez les résultats donnés par ces différentes méthodes.

II Application d'une transformation spatiale sur une image

II.1 Changement d'échelle: Interpolation au plus proche voisin

L'interpolation au plus proche voisin consiste à attribuer à chaque pixel de l'image d'arrivée la valeur du pixel dans l'image source le plus proche de son antécédent par la transformation inverse. C'est une interpolation polynomiale d'ordre 0.

Compléter la fonction **zoom_image_ppv()** permettant de zoomer dans une image reconstruite à l'aide d'une interpolation au plus proche voisin. Cette fonction prend comme arguments: une matrice stockant l'image originale à agrandir (**image_in**), une matrice stockant l'image agrandie que vous allez générer (**image_out**), les nombres de lignes et de colonnes des matrices images (respectivement **width** et **height**), la variable réelle **zoom_factor** (qui détermine le facteur de changement d'échelle à appliquer).

On note (x_{in}, y_{in}) (respectivement (x_{out}, y_{out})) les coordonnées des pixels dans l'image d'entrée (respectivement l'image de sortie). La fonction **zoom_image_ppv()** parcourra l'ensemble des pixels de l'image de sortie (x_{out}, y_{out}) et calculera leurs coordonnées dans l'image d'entrée (x_{in}, y_{in}) de la manière suivante:

$$(x_{in}, y_{in}) = \left(\frac{x_{out} - \frac{\text{width}}{2}}{\text{zoom\_factor}} + \frac{\text{width}}{2}, \frac{y_{out} - \frac{\text{height}}{2}}{\text{zoom\_factor}} + \frac{\text{height}}{2} \right)$$

Tester votre routine sur **lena.pgm** lorsque **zoom_factor=10** et visualisez avec **The Gimp** le résultat obtenu.

II.2 Changement d'échelle: Interpolation bilinéaire

L'interpolation bilinéaire consiste à attribuer à chaque pixel interpolé une combinaison linéaire des quatre pixels les plus proches. C'est une interpolation polynomiale d'ordre 1.

Complétez la fonction **interpolation_bilineaire()** permettant de zoomer dans une image reconstruite à l'aide d'une interpolation bilinéaire. Cette fonction prend en entrée: une matrice stockant l'image à interpoler (**image**), les nombres de lignes et de colonnes de la matrice image (respectivement **width** et **height**), les variables réelles **x** et **y** (qui déterminent les coordonnées du pixels à interpoler). La valeur du niveau de gris interpolée sur les coordonnées réelles (x, y) est stockée dans la variable **interpolation_bilineaire** puis retournée en sortie de la fonction.

Complétez la fonction **zoom_image_bl()** permettant de zoomer dans une image reconstruite à l'aide d'une interpolation bilinéaire (les arguments d'entrée/sortie sont identiques à ceux de la fonction **zoom_image_ppv()** détaillés dans la section II.1). Comparez avec **The Gimp** la qualité d'image obtenue pour **zoom_factor=10** lorsque l'interpolation bilinéaire est utilisée.

II.3 Translation horizontale et verticale

Complétez la fonction **translate_image()** appliquant une translation "sub-pixel" sur une image. Cette fonction prend comme arguments: une matrice stockant l'image originale à translater (**image_in**), une matrice stockant l'image translatée que vous allez générer (**image_out**), le nombre de lignes et de colonnes des matrices images (respectivement **width** et **height**), un tableau de nombres réels de taille 2 (qui stocke les valeurs de translation horizontale et verticale à appliquer).

III Recalage d'images à l'aide d'une minimisation au sens des moindres carrés

On dispose de 2 images: une image de référence (I_1) et une image à recaler (I_2). On note $I_1(x, y)$ la valeur du niveau de gris de I_1 au pixel (x, y). On note également $p = \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} = \begin{pmatrix} \Delta X \\ \Delta Y \end{pmatrix}$ le vecteur des paramètres de translation recherchés.

L'algorithme consiste à translater I_2 de p puis à calculer $D(p) = I_1 - \text{Translaté}(I_2, p)$. $D_{x,y}(p)$ est donc la fonction décrivant la différence entre l'image de référence et l'image à recaler au pixel (x, y) lorsque le vecteur des paramètres transformation prend la valeur p. L'objectif est de trouver p tel que $D(p) = 0$. Pour cela, on initialise p à $(0, 0)$, on calcule $D(p)$, puis on cherche $t = \begin{pmatrix} t_1 \\ t_2 \end{pmatrix}$ tel que $D(t + p) = 0$ à l'aide d'une approximation de Taylor au premier ordre de la manière suivante:

$$D_{x,y}(p + t) = D_{x,y}(p) + t_1 \frac{\partial D_{x,y}(p)}{\partial p_1} + t_2 \frac{\partial D_{x,y}(p)}{\partial p_2} + ...$$

L'approximation de Taylor fournit ainsi le système surdéterminé à résoudre suivant:

$$\begin{pmatrix} -\frac{\partial D_{1,1}(p)}{\partial p_1} & -\frac{\partial D_{1,1}(p)}{\partial p_2} \\ -\frac{\partial D_{2,1}(p)}{\partial p_1} & -\frac{\partial D_{2,1}(p)}{\partial p_2} \\ . & . \\ . & . \\ . & . \end{pmatrix} \begin{pmatrix} t_1 \\ t_2 \end{pmatrix} \approx \begin{pmatrix} D_{1,1}(p) \\ D_{2,1}(p) \\ . \\ . \\ . \end{pmatrix}$$

Pour simplifier les équations, on note:

$$A = \begin{pmatrix} -\frac{\partial D_{1,1}(p)}{\partial p_1} & -\frac{\partial D_{1,1}(p)}{\partial p_2} \\ -\frac{\partial D_{2,1}(p)}{\partial p_1} & -\frac{\partial D_{2,1}(p)}{\partial p_2} \\ . & . \\ . & . \\ . & . \end{pmatrix} \qquad B = \begin{pmatrix} D_{1,1}(p) \\ D_{2,1}(p) \\ . \\ . \\ . \end{pmatrix}$$

En utilisant l'approximation au sens des moindres carrés, l'estimation des paramètres peut donc être effectuée de la manière suivante:

$$t = (A^T A)^{-1} A^T B$$

III.1 Recalage d'un mouvement de faible amplitude

A l'aide des routines écrites dans le paragraphe II.3, simulez à partir de l'image originale **lena.pgm** une image translatée horizontalement de 1 pixels seulement. Estimez cette transformation à l'aide d'un recalage au sens des moindres carrés. Il est nécessaire pour cela d'inverser la matrice $A^T A$. Au lieu de l'inverser, on la factorisera (sous forme LU) avec la sous-routine **dgetrf()**, comme vu précédemment au TP2. La résolution du système linéaire se fera avec **dgetrs()**. On rappelle que, pour compiler le fichier Fortran, on n'oubliera pas d'inclure Lapack et Blas dans ligne de compilation de la manière suivante:

```
gfortran -o run tp6.f90 -llapack -lblas
```

III.2 Recalage d'un mouvement de forte amplitude

Simulez une image translatée horizontalement de 50 pixels. L'estimation du mouvement est-elle satisfaisante? A cause des effets de bord sur l'image, l'approximation de Taylor décrite précédemment n'est valide que pour de petites translations. Le t calculé au paragraphe III.1 ne vérifie donc pas exactement $D(t+p) = 0$. On pose donc $t_0 = t$ et $p_0 = p$, et $p_1 = t_0 + p_0$. On cherche ensuite t_1 tel que $D(p_1 + t_1) = 0$ et on se retrouve donc dans la situation de départ. On calcule ainsi une suite p_n et l'estimation des paramètres peut donc être effectuée à l'aide du schéma itératif suivant:

$$p_{n+1} = p_n + t_n = p_n + (A^T A)^{-1} A^T B$$

Implémentez ce schéma itératif. A chaque itération n, la norme 2 de t_n sera calculée. Le schéma itératif sera arrêté dès lors que $\|t_n\|_2$ se trouve inférieure à $\epsilon = 10^{-14}$. Vérifiez que le déplacement horizontal de 50 pixels est maintenant correctement estimé.

III.3 Résolution numérique

Implémentez et comparez les performances en terme de temps de calculs des différentes méthodes vues en cours pour la résolution en t du système $(A^T A)t = A^T B$.

IV Optionnel: Recalage affine

Modifiez votre code afin que l'algorithme puisse simuler puis estimer une transformation spatiale affine.