TS223 - Introduction au traitement d'images
Travaux Pratiques

Rémi Giraud
remi.giraud@enseirb-matmeca.fr
2024-2025

Espaces de couleur

Chroma-Keying

Le "chroma-keying" ou "incrustation en chrominance" est une technique de fusion d'images en couleurs. La méthode consiste à isoler puis remplacer les pixels "de fond" d'une image, de couleur caractéristique, par les pixels correspondants d'une seconde image (voir Figure 1). Un exemple type est la carte météorologique incrustée en arrière-plan d'un présentateur alors que celui-ci est filmé sur un fond de couleur verte ou bleue.

Figure 1: images sources (arrière-plan et avant-plan) et image fusionnée.

La fusion est opérée à l'aide d'un masque binaire M associé à l'image d'avant-plan, selon l'expression suivante :

Masque binaire M

$$I_{\text{fusion}} = \begin{cases} I_{\text{avant}} & \text{si} \quad M = 1 \\ I_{\text{arriere}} & \text{si} \quad M = 0 \end{cases}$$

Dans le cas d'un arrière-plan de couleur caractéristique bleue, l'application de la technique "chroma-keying" peut-être facilitée par l'utilisation de l'espace de représentation de couleurs YCbCr au lieu du classique espace RGB.

Travail demandé :

- ☐ Observez et commentez les différentes composantes RGB et YCbCr de l'image *pool.tif*, plus précisément pour les régions rouges, bleues et blanches. Quel semble être l'intérêt de la représentation de couleurs YCbCr ?

- ☐ Réalisez la fusion des images *people.jpg* et *metro.jpg*. L'objectif est dans cet exemple de remplacer le fond de *people.jpg* : le bleu est par conséquent la couleur caractéristique sur laquelle le masque sera construit par seuillage des intensités. Vous pouvez également utiliser les images *foreground.jpg* et *background.jpg*.

- ☐ Commentez la pertinence de la représentation YCbCr par rapport à la représentation RGB pour cette application en observant les canaux B (de RGB) et Cb (de YCbCr).

Rappel : Fonction pour changer d'espace : `skimage.color.rgb2ycbcr`

Détection de tâches

Travail demandé :

- ☐ Segmentez (c-à-d, obtenir un masque `M` détectant) les tâches saturées blanches sur l'image de motion capture *mocap.jpg* en seuillant les intensités sur la luminance de l'image.

- ☐ Faire un étiquetage en composantes connexes `L` de l'image binaire obtenue :

```
L = skimage.measure.label(M) ;
```

- ☐ Affichez l'image étiquetée en utilisant une palette de couleur qui met en évidence leur numérotation :

```
map_ = np.random.rand(np.max(L)+1, 3)
map_1 = np.ones((np.max(L)+1,1))
palette = np.concatenate((map_, map_1), axis=1)
```

- ☐ Superposez les tâches détectées à l'image initiale

(Attention aux intensités de l'image et `Lrgb`) :

```
L_rgb = (palette[L, 0:3]*255).astype('uint8')
...
```

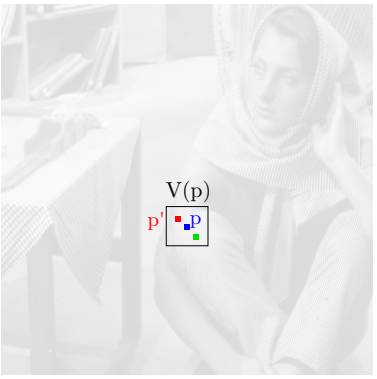
Filtrage bilatéral

L'objectif de cette partie est d'implémenter le filtre bilatéral qui combine filtrage spatial et couleur et pourra être appliqué au débruitage (*barbara_awgn_noise.png*) et au lissage d'images (*face.png*).

1. Principe du filtre

Ce filtrage consiste pour un pixel à traiter p dans une image I, à agréger l'information de ses pixels voisins p' dans un voisinage régulier $V(p)$, selon :

$$I_F(p) = \frac{\sum_{p' \in V(p)} w(p,p') I(p')}{\sum_{p' \in V(p)} w(p,p')}.$$



Ici les intensités de l'image I aux pixels p' contribuent à déterminer l'intensité du pixel p dans l'image filtrée I_F selon leur proximité au pixel p exprimée par le poids $w(p,p')$. A noter que si $w(p,p') = 1$, on retrouve l'expression d'un filtre moyenneur uniforme.

Dans le filtrage bilatéral, ce poids est composé de deux termes, un poids spatial w_s qui exprime la proximité spatiale des pixels aux positions $p = [i,j]$ et $p' = [i',j']$, et un poids couleur w_c qui exprime la proximité entre les couleurs des pixels $I(p)$ et $I(p')$ afin de pondérer la contribution des pixels similaires dans le voisinage :

$$w(p,p') = w_s(p,p') w_c(p,p') = \exp\left(-\frac{\|p-p'\|_2}{2\sigma_s^2}\right) \exp\left(-\frac{\|I(p)-I(p')\|_2}{2\sigma_c^2}\right),$$

avec σ_s et σ_c des paramètres que l'on fixera de manière empirique selon l'application. Un exemple de lissage obtenu avec ce filtrage est donné en Figure 2.

Image initiale | Image filtrée

Figure 2: Exemple de résultat du filtre bilatéral.

Travail demandé :

☐ Implémentez le filtre bilatéral depuis cette base de code :

```
img = plt.imread('../img/barbara_awgn_noise.png').astype('double')
h, w = img.shape
#Paramètres (à faire varier)
v_size = 5
sigma_s = 3
sigma_c = 1
img_bf = img*0
#Parcours de tous les pixels de l'image
for i in range(0,h):
  print(i)
  for j in range(0,w):
    sum_w = 0
    #Sélection d'une fenêtre (2*v_size+1)x(2*v_size+1) ajustée
    for ii in range(max(i-v_size, 0), min(i+v_size, h)):
      for jj in range(max(j-v_size, 0), min(j+v_size, w)):
      #...
```

☐ Appliquez le aux images *barbara_awgn_noise.png* pour le débruitage et *face.png* pour un effet de lissage.

☐ Quel est l'impact de chaque paramètre : σ_s, σ_c, v_size ?

☐ Vous pouvez tester l'algorithme sur vos images personnelles...

Filtrage fréquentiel

L'objectif de cette partie est d'abord d'identifier, dans l'espace des fréquences, la signature d'une trame visible sur une image (*pise_ext.bmp*), puis de générer un filtre linéaire RIF (Réponse Impulsionnelle Finie) adapté permettant de l'atténuer.

Travail demandé :

☐ a) Chargez l'image *pise_ext.bmp*. Cette image est perturbée par du bruit haute-fréquences additif.

Affichez l'image bruitée ainsi que sa TF.

Repérez les fréquences de bruit.

☐ Débruitez l'image à l'aide d'un filtre passe-bas (ex. moyenneur uniforme)

Affichez l'image débruitée, sa TF et la différence entre l'image débruitée et l'image bruitée. Le filtrage a-t-il été efficace ?

On se propose à présent de réaliser un filtre "coupe-bande" (qui coupe l'information sur une certaine bande de fréquence) de type gaussien dans le domaine fréquentiel. Il faudra donc réfléchir à comment créer un filtre, qui sera construit et appliqué dans le domaine spatial (convolution à l'image), mais dont la transformée de Fourier aura l'aspect attendu d'un filtre coupe bande (1 partout et deux "creux" aux pics correspondant au bruit).

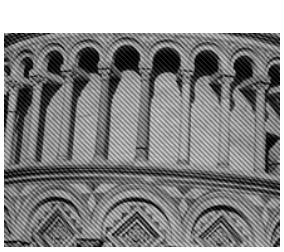
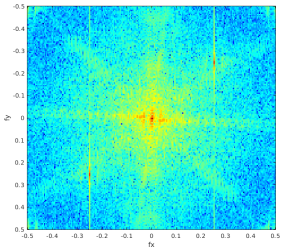
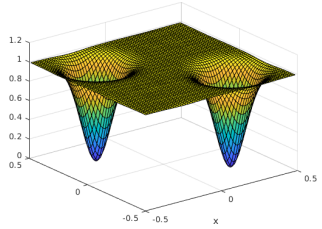


Image bruitée



TF image bruitée



Filtre fréquentiel coupe bande G_{cbande}

Figure 3: Bruit fréquentiel et filtre coupe bande.

Pour rappel, une convolution dans le domaine spatial correspond à une multiplication dans le domaine fréquentiel. En convoluant par notre filtre spatial dont la TF est un filtre coupe-bande, on débruitera bien l'image. Pour créer ce filtre "coupe-bande" fréquentiel, on procédera en trois étapes :

1)- On créera d'abord un filtre "passe-bas" défini par une fonction gaussienne bidimensionnelle centrée, isotrope :

$$g_{\text{pbas}}(x, y) = \frac{1}{2\pi\sigma^2} \exp^{-\frac{x^2+y^2}{2\sigma^2}}$$

Pour rappel les cartes x, y, peuvent être obtenues en utilisant `X, Y = np.meshgrid(range(-size,size+1), range(-size,size+1)`

Pour rappel dans l'espace fréquentiel, une Gaussienne de variance σ donne aussi une Gaussienne mais de variance $1/\sigma$.

2)- Ce filtre sera ensuite modulé par un signal de type sinusoïdal de sorte à le centrer sur la fréquence parasite correspondant au bruit, pour en faire un filtre “passe-bande” :

$$g_{\text{pbande}}(x, y) = g_{\text{pbas}}.2.cos(2.pi.(fx.x + fy.y)).$$

Multiplier par un cosinus dans le domaine spatial, va en effet correspondre à une convolution de la TF de g_{gpbas} par la TF du cosinus, c'est à dire deux Dirac aux fréquences portées.

3)- Une dernière opération, permettra dans le domaine fréquentielle de transformer ce filtre en un filtre coupe-bande :

$$g_{\text{cbande}} = \text{Dirac} - g_{\text{pbande}}.$$

En effet, pour un Dirac (impulsion centrée en 0, c'est à dire un 1 au milieu de votre matrice pleine de zéros), la TF vaut 1 partout. On aura donc un filtre fréquentiel coupe-bande G_{cbande} :

$$G_{\text{cbande}} = 1 - G_{\text{pbande}}.$$

Travail demandé :

- ☐ Construisez le filtre à chaque étape et visualiser son aspect dans les domaines spatial et fréquentiel.
- ☐ Comment doit-on choisir σ de sorte à respecter le théorème de Shannon?
- ☐ Commentez le résultat de l'application du filtre final sur l'image tramée. Précisez notamment l'influence des différents paramètres.

Transformations spatiales

Objectif : Manipuler les modèles de transformation géométrique. Effectuer des transformations géométriques d'image selon différents modèles.

Rotation d'une image

Objectif : Appliquer à l'image *cameraman.tif* une rotation de 45^o de centre de rotation le milieu de l'image.

Puisque l'on veut remplir de manière dense (tous nos pixels) notre image finale, toujours dans l'espace rectangulaire de l'image initiale (h×w). On doit donc calculer la transformation inverse pour savoir où piocher les intensités dans l'image initiale. Donc pour faire une rotation de 45^o dans un repère standard qui serait orienté droite(x)-haut(y) (voir Figure 4), on doit donc effectuer une rotation de -45^o dans ce même repère. Si on garde le repère Matlab centré en haut à gauche et d'orientation droite(x)-bas(y), faire une rotation de 45^o revient à faire cette transformation de -45^o. Les positions des pixels ayant subi la rotation étant flottantes, la fonction `interp2` doit être appelée pour moyenner les valeurs des pixels les plus proches.

Travail demandé :

- ☐ Créez un pavage vertical et horizontal X,Y de la taille de l'image (`np.meshgrid`).
- ☐ Déterminez la matrice de rotation `R` selon l'angle de rotation.
- ☐ Appliquez la rotation du pavage par rapport au centre de l'image.
- ☐ Calculez l'interpolation en utilisant la fonction `scipy.interpolate.RectBivariateSpline`.

Image initiale *cameraman.tif*

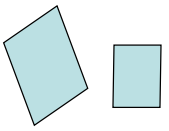
Rotation de 45^o

Figure 4: Exemple de rotation discrète avec interpolation

Objectif : Manipuler les modèles de transformation géométrique. Effectuer des transformations géométriques d'image selon différents modèles.

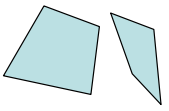
On peut définir un modèle générique de transformation affine de la manière suivante :

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$



Et un modèle de transformation homographique ainsi :

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$



La fonction suivante applique une transformation homographique de paramètres H à une image I, afin de produire une image J de taille $s = [Jh, Jw]$.

```
def transformimage(I, H, s):
  Jh = s[0]
  Jw = s[1]
  Ih = I.shape[0]
  Iw = I.shape[1]
  X2, Y2 = np.meshgrid(range(0,Jw), range(0,Jh))
  invH = np.linalg.inv(H)
  X = np.zeros((Jh,Jw))
  Y = np.zeros((Jh,Jw))
  for i in range(0, Jh):
    for j in range(0, Jw):
      P2 = [[X2[i,j]], [Y2[i,j]], [1]]
      P = invH@P2
      X[i,j] = P[0]/P[2]
      Y[i,j] = P[1]/P[2]
  x = np.arange(0, Iw)
  y = np.arange(0, Ih)
  spl = RectBivariateSpline(y, x, I, kx=2, ky=2)
  J = spl.ev(Y, X).astype(np.float32)
  #Mise a zero des pixels en dehors de l'image
  for i in range(0, Jh):
    for j in range(0, Jw):
      if ( (Y[i,j]<0) or (Y[i,j]>Ih) or (X[i,j]<0) or (X[i,j]>Iw)):
        J[i,j] = 0
  return J
```

Transformation affine

a) Charger l'image *lena256.png*. Appliquez cette fonction afin de générer une translation de vecteur $dx = +10$, $dy = +20$, qui correspond à la matrice d'homographie suivante :

```
H = [[1, 0, 10], [0, 1, 20], [0, 0, 1]]
```

Afficher le résultat.

b) Appliquer la fonction pour un changement d'échelle autour du point $(x0, y0)$, correspondant à la matrice H suivante :

```
x0 = 100
y0 = 100
s = 0.8 # facteur d'échelle
HT = np.array([[1,0,x0], [0,1,y0], [0,0,1]])
HS = np.array([[s,0,0], [0,s,0], [0,0,1]])
H = HT @ HS
```

c) Appliquer la fonction pour une rotation autour du point $(x0, y0)$, correspondant à la matrice H suivante :

```
x0 = 100
y0 = 100
a = np.pi/180* 70; # 20 degrees
HT = np.array([[1,0,x0], [0,1,y0], [0,0,1]])
HR = np.array([[np.cos(a),np.sin(a),0], [-np.sin(a),np.cos(a),0],
[0,0,1]])
H = HT @ HR @ np.linalg.inv(HT)
```

Transformation homographique

a) Notons `pts1` les coordonnées des coins de l'image source (Attention : on suppose les coins dans l'ordre haut-gauche, haut-droit, bas-gauche, puis bas-droit) :

```
Ih, Iw = I.shape
pts1 = np.array([[1,1], [Iw,1], [1,Ih], [Iw,Ih]])
plt.figure(4)
plt.imshow(I)
plt.title('Image initiale')
plt.plot(pts1[:,0], pts1[:,1], linewidth=2)
```

b) On souhaite transformer l'image par une transformation homographique afin de ramener les quatre coins en des positions choisies par l'utilisateur.

Notons `pts2` les coordonnées correspondantes dans l'image destination (de taille $Jh \times Jw = 256 \times 256$). On peut obtenir ces coordonnées à partir de la souris grâce à `ginput`.

c) Extraire les paramètres de transformation géométrique sous la forme d'une matrice d'homographie à l'aide de la fonction `findHomography` :

```
def findHomography(src_points, dst_points):
  A = []
  for i in range(0, 4):
    x = src_points[i,0]
    y = src_points[i,1]
    u = dst_points[i,0]
    v = dst_points[i,1]
    A.append([x, y, 1, 0, 0, 0, -u*x, -u*y, -u])
    A.append([0, 0, 0, x, y, 1, -v*x, -v*y, -v])
  A = np.array(A)
  U, S, V = np.linalg.svd(A)
  H = V[-1].reshape(3, 3)
  return H / H[2, 2]
```

Effectuer la transformation d'image de la façon suivante :

```
H = findHomography(pts1, pts2)
J = transformimage(I, H, [Jh, Jw])
```

Afficher l'image J obtenue, et lui superposer les points `pts2`.

Extra : Applications

a) Les codes-barres matricels (DataMatrix, QRCode...) sont un outil particulièrement simple et efficace pour étiqueter un objet ou coder une information numérique (comme une URL) sur un support papier. En pratique, l'apparence du code-barre dans une image capturée par un appareil photo ou une caméra est modifiée en fonction du point de vue. La première étape avant de pouvoir décoder un tel code-barre est donc de le recaler, *c-à-d.* transformer l'image pour que le code prenne une place prédéfinie dans une image de taille connue.

Utiliser la technique précédente pour extraire le code-barre recalé à partir des images fournies.

Attention : dans ce cas, le rôle de `pts1` et `pts2` est inversé. `pts1` joue ainsi le rôle de points dans l'image d'origine situés sur les coins du motif déformé, `pts2` correspond aux coins dans l'image de destination :

```
pts1 = [x, y]
Jh = 256
Jw = 256
pts2=[[1,1], [Jw,1], [1,Jh], [Jw,Jh]]
```

Pour l'image "qr-code-wall.jpg", on fournit les coordonnées dans l'image d'origine (on pourra utiliser `ginput` pour les autres) :

```
x = [53, 275, 62, 275]
y = [48, 49, 264, 262]
```

b) Application à la réalité augmentée.

Une autre application possible est d'insérer une image 2D dans l'image représentant un monde 3D. En reprenant l'exemple du code barre précédent, insérer l'image de Lena à la place du code barre dans l'image initiale (voir Figure 2). Faire attention à ce que les tailles d'images soient compatibles. Indication : le masque de composition (cf. chroma-key) peut-être obtenu en transformant une image de la même taille que Lena et contenant uniquement des 1.

Figure 5: Exemples : (en haut) extraction d'une image par transformation spatiale, (en bas) insertion et composition d'une image.