IT220 - Traitement d'images

Travaux Pratiques

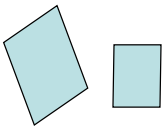
Rémi Giraud
remi.giraud@enseirb-matmeca.fr
2023-2024

Exercice 5 : Transformations géométriques

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

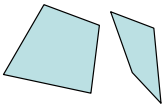
On peut définir un modèle générique de transformation affine de la manière suivante :

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



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]
  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, s[0]):
    for j in range(0, s[1]):
      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]
  J = np.zeros((Jh, Jw))
  x = np.arange(0, Jw)
  y = np.arange(0, Jh)
  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, s[0]):
    for j in range(0, s[1]):
      if ( (Y[i,j]<0) or (Y[i,j]>Jh) or (X[i,j]<0) or (X[i,j]>Jw)):
        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 1: Exemples : (en haut) extraction d'une image par transformation spatiale, (en bas) insertion et composition d'une image.