SY32 – TD Vision 03 : Transformations géométriques

Exercice 1 : Transformations affines

La fonction suivante applique une transformation homographique de paramètres H à une image I, afin de produire une image O de taille $[h, w]$:

```
import numpy as np
from scipy.interpolate import griddata

def transformer(I, H, hw = (-1,-1), interp='linear'):
    h = hw[0]
    w = hw[1]
    if (w <= 0 or h <= 0):
        (h,w) = hw = I.shape[:2]
    O = np.zeros((h,w)) # image de sortie

    xx1, yy1 = np.meshgrid(np.arange(I.shape[1]), np.arange(I.shape[0]))
    xx1 = xx1.flatten()
    yy1 = yy1.flatten()

    Hinv = np.linalg.inv(H)

    xx2, yy2 = np.meshgrid(np.arange(O.shape[1]), np.arange(O.shape[0]))
    xx2 = xx2.flatten()
    yy2 = yy2.flatten()

    xxyy2 = np.stack((xx2,yy2,np.ones((O.size))), axis=0)

    xxyy = Hinv @ xxyy2
    xxyy = np.stack((xxyy[0]/xxyy[2], xxyy[1]/xxyy[2]), axis=0)

    O = griddata((xx1,yy1), I.flatten(), xxyy.T, method=interp, fill_value=0).reshape(O.shape)

    return O
```

1. Charger l'image `skimage.data.camera`. Utiliser cette fonction afin d'opérer sur cette image une translation de vecteur ($T_x = 10$, $T_y = 20$), qui correspond à la matrice d'homographie :

$$\mathsf{H} = \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix}$$

   Afficher le résultat. Une illsutration du résultat est donnée en Figure 1.

FIGURE 1 – Translation de l'image Camera.

2. Appliquer maintenant un changement d'échelle de facteur $r = 0,4$ dans les deux directions et autour du point $(x_0 = 100, y_0 = 100)$. La matrice d'homographie correspondante est :

$$\mathsf{H} = \begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} r & 0 & 0 \\ 0 & r & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} 1 & 0 & -x_0 \\ 0 & 1 & -y_0 \\ 0 & 0 & 1 \end{bmatrix}$$

   Une illsutration du résultat est donnée en Figure 2.

FIGURE 2 – Changement d'échelle de l'image Camera autour du point (x_0, y_0).

3. De la même manière, appliquer une rotation d'angle $\alpha = 20$ degrés autour du point ($x_0 = 100, y_0 = 100$). Sachant que la matrice de rotation s'écrit $\mathsf{R} = \begin{bmatrix} cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{bmatrix}$, quelle est l'expression de la matrice H pour la transformation demandée ?
Une illsutration du résultat est donnée en Figure 3.

FIGURE 3 – Rotation de l'image Camera autour du point (x_0, y_0).

Exercice 2 : Homographie

On souhaite transformer une image de sorte à la ramener à 4 coins définis par l'utilisateur. Pour cela il faut estimer la matrice de transformation homographique pour passer de l'image d'origine à l'image transformée.

1. Nous allons transformer l'image `skimage.data.camera`. Définir un tableau $\mathbf{coins}_I$ contenant les coordonnées de ses 4 coins, dans l'ordre haut-gauche, haut-droit, bas-droit, puis bas-gauche, comme suit :
$$\mathbf{coins}_I = \begin{bmatrix} x_{hg} & y_{hg} \\ x_{hd} & y_{hd} \\ x_{bd} & y_{bd} \\ x_{bg} & y_{bg} \end{bmatrix}$$
(Rappel : pour une image, l'origine est située en haut à gauche.)
2. Nous allons incruster cette image dans le cadre visible sur le mur à l'arrière plan de l'image oggy_salon.jpg (voir Figure 4).
Ouvrir oggy_salon.jpg, qui sera l'image de destination. L'afficher, et permettre à l'utilisateur de sélectionner quatre points en utilisant la fonction `matplotlib.pyplot.ginput`. Les points sélectionnés vont indiquer les coordonnées des 4 coins de l'image à transformer après transformation. Les placer dans un tableau $\mathbf{coins}_O$ selon le même ordre que dans $\mathbf{coins}_I$. Pour l'incrutation, ces points devraient correspondre aux coins du cadre dans oggy_salon.jpg.
3. Extraire les paramètres de transformation géométrique sous la forme d'une matrice d'homographie H à l'aide de la fonction `skimage.transform.estimate_transform` et des 4 paires de points correspondants entre $\mathbf{coins}_I$ et $\mathbf{coins}_O$, de la façon suivante :

```
tform = transform.estimate_transform('projective', coinsI, coinsO)
H = tform.params
```

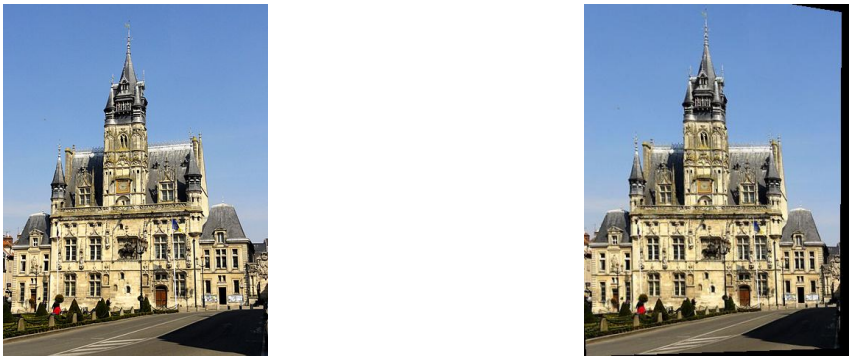
4. Utiliser cette matrice pour effectuer la transformation de l'image camera à l'aide de la fonction `transformer` donnée dans l'Exercice 1. Fixer les dimensions de l'image de sortie avec le

FIGURE 4 – Image oggy_salon.jpg, dans laquelle on souhaite modifier le cadre par une autre image.

paramètre `hw`, de telle sorte à ce qu'elles correspondent aux dimensions de l'image où faire l'incrustation. Afficher l'image transformée sur fond noir, en lui superposant les points choisis dans la question 2.

5. Incruster l'image transformée dans la scène oggy_salon.jpg. Définir et utiliser un masque peut-être nécessaire pour parvenir à faire l'incrstation.
   Attention : l'image camera a 1 seul canal pour les niveaux de gris, tandis que l'image oggy_salon.jpg a 3 canaux pour les couleurs.

bonus Correction de la perspective d'un plan.
En exploitant et modifiant le travail effectué, trouver comment corriger l'effet perspective sur le bâtiment de l'image 360px-Comp.jpg illustrée en Figure 5.



(a) Photo de l'Hôtel de ville de Compiègne. (b) Correction de la perspective sur le mur frontal.

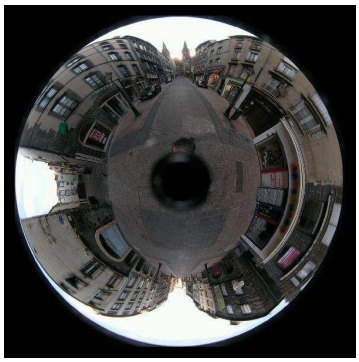
FIGURE 5 – Image 360px-Comp.jpg dont la perspective est à corriger pour observer le bâtiment de manière parfaitement frontale.

Exercice 3 : Transformation en coordonnées polaires

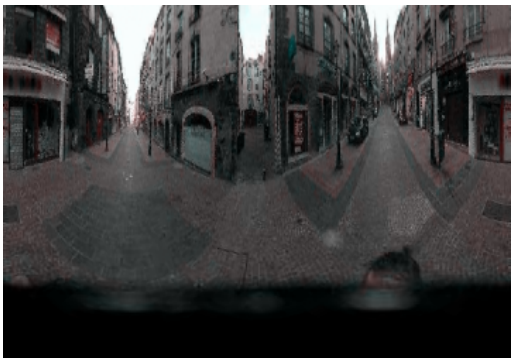
On souhaite transformer l'image omnidirectionnelle de la Figure 6 (oldtown.png) en image panoramique par représentation du contenu de l'image en coordonnées polaires. L'image d'origine est de dimensions 600 × 600 pixels. Le rayon du disque image est de 250 pixels.

1. Charger l'image oldtown.png et la convertir en niveaux de gris pour faciliter les traitements (fonction `skimage.color.rgb2gray`). Définir x_0 et y_0 les coordonnées de son centre. Créer deux matrices xx1 et yy1 représentant les coordonnées dans cette image (fonction `numpy.meshgrid`).
2. Créer le canevas de l'image de destination, l'initialiser avec toutes les valeurs à zéro et de dimensions 250 lignes (représentent le *rayon* dans l'image d'origine) pour 360 colonnes (représentent l'*angle*). Créer deux matrices xx2 et yy2 représentant les coordonnées dans l'image de destination (fonction `numpy.meshgrid`).
   Dans l'image de destination, xx2 et yy2 vont représenter les coordonnées polaires de l'image source : xx2 l'angle (entre 0 et 360 degrés) et yy2 le rayon par rapport au centre (x_0, y_0) (entre 0 et 250 pixels).
3. Calculer les coordonnées polaires de tous les points de l'image source (à partir des matrices xx1 et yy1, veiller à fixer les coordonnées du centre à $(x_0, y_0) = (0, 0)$).

1. Issue de « Toward Flexible 3D Modeling using a Catadioptric Camera », M. Lhuillier, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Minneapolis, USA, June 2007.



(a) Image omnidirectionnelle (catadoptrique) [1].



(b) Image après transformation en coordonnées polaires.

FIGURE 6 – Image oldtown.png à transformer en vue panoramique par transformation en coordonnées polaires.

Ensuite, reprojeter l'intensité de ces points dans l'image de destination, à la position donnée par les coordonnées polaires (arrondies à une valeur entière dans les limites de la zone de l'image). Veiller à ne considérer que les points du disque image, c'est-à-dire dont le rayon < 250 pixels. Commenter le résultat.

4. La méthode précédente génère des trous et des artefacts. Pour appliquer correctement la transformation, il faut chercher les coordonnées dans l'image source à partir des points de l'image de destination, et en interpoler les intensités (positions sous-pixelliques). Reprendre tout à zéro et développer la méthode par transformation inverse :
   (a) À partir des coordonnées (xx2,yy2), calculer les coordonnées correspondantes (xx,yy) dans l'image source « oldtown » (**en chiffres flottants**, en vue de faire l'interpolation).
   (b) Utiliser `scipy.interpolate.griddata` pour effectuer l'interpolation des valeurs de oldtown.png aux positions xx,yy et l'intégrer dans l'image de destination, puis afficher l'image obtenue. (S'insiper de la fonction `transformer` donnée dans l'Exercice 1.)
5. Appliquer le même traitement à chaque composante R,V,B de l'image d'origine afin d'obtenir une image transformée en couleurs.

Liste de fonctions utiles :

- `skimage.io.imread`
- `skimage.util.img_as_float`
- `numpy.copy`
- `numpy.stack`
- `numpy.pi`
- `numpy.flip`
- `numpy.array`
- `numpy.identity`
- `numpy.deg2rad`
- `numpy.rad2deg`
- `numpy.matmul` (ou opérateur `@`)
- `numpy.sqrt`
- `numpy.cos`
- `numpy.sin`
- `numpy.arctan2`
- `numpy.logical_and` (ou opérateur `&`)
- `numpy.arange`
- `numpy.meshgrid`
- `numpy.griddata`
- `numpy.flatten`
- `nupy.reshape`