TD1 – Calibration et Matrice intrinsèque – 3 séances

Objectif :

- **Calibrer le système à partir d'une image de référence**
- **Calcul les coordonnées 3D des coins du damier dans l'image de référence**

1. Calibration : Matrice intrinsèque

1.1 Estimation manuelle

On rappelle que les paramètres intrinsèques (matrice de calibration) sont les coefficients de la matrice K suivante :

$$K = \begin{bmatrix} \alpha_x & 0 & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix}$$

On considère l'image de référence fournie, de taille 960 x 540 pixels (colonnes x lignes), contenant un damier. L'acquisition de cette image a été réalisée avec le damier placé perpendiculairement à l'axe optique de la caméra et à une distance (approximative) de 500 mm de la caméra. La taille réelle du damier (5x8 cases régulières) est de 155 mm par 105 mm. On considère que le repère caméra est tel que l'axe des X est selon les colonnes de l'image et l'axe des Y est selon les lignes de l'image. On considère que le bord horizontal du damier est parfaitement aligné (parallèle) à l'axe des X (il en est de même pour le bord vertical par rapport à l'axe des Y).

L'objectif de l'exercice est de déterminer les paramètres intrinsèques de la caméra (calcul manuel sur feuille). Pour réaliser cette estimation, on mesurera les dimensions appropriées du damier en pixel, avec un logiciel permettant d'ouvrir une image numérique et mesurer des distances (par exemple Gimp). On fera l'hypothèse que l'axe optique passe par le centre de l'image numérique (ce qui permet de déduire x0 et y0).

Indication : en supposant que dX=155 mm est une distance selon l'axe des X (horizontal dans notre cas), on pourra considérer que cette distance dX est la différence (selon les X) entre les coordonnées X1 et X2 de deux points P1 et P2 tel que : dX=X2-X1. Chacun des points P1 et P2 conduisent à des points p1 et p2 dans l'image numérique tel que p1=K*P1, et tel que la différence des coordonnées selon x (dans le repère de l'image numérique) dx=x2-x1 est mesurable en pixels... On se retrouve alors avec un système d'équation « simple » à résoudre.

1.2 Matrice intrinsèque en Python (numpy.array) : fonction helper.create_K()

Implémenter le calcul en de cette matrice en Python : créer la matrice K (numpy array) et affecter ses coefficients en fonction des données initiales (dx, dy, Z, taille image, dX, dY).

Remarque 1: pour créer un tableau numpy de n lignes et m colonnes initialisé à 0, on peut utiliser : « numpy.zeros((n,m)) ».

Remarque 2: Pour les exercices suivants, on placera le code de création de la matrice K dans une fonction (e.g. create_K) dans un fichier (e.g. helper.py) que l'on placera dans le répertoire de travail (utilisation par : import helper ; K=helper.create_K()).

Votre script invoquant la fonction [create_K()] aura la forme suivante :

```
import numpy as np
import helper

K=helper.create_K()
print(K)
```

2. Matrice intrinsèque et changement de repère « pixels 2D ↔ caméra 3D »

- Fonction 1 : calculer les coordonnées 3D dans le repère caméra depuis des coordonnées « pixels » 2D, en connaissant leur côte (Z) et la matrice de calibration K.
- Fonction 2 : calculer les coordonnées 2D (« pixels ») depuis des coordonnées 3D exprimées dans le repère caméra, en connaissant la matrice de calibration K.

2.1: Première étape « manuelle »

L'objectif est de déterminer, dans le contexte précédent (image de référence, avec Z=500 mm connu), les coordonnées 3D (X,Y,Z) dans le repère de la caméra des coins des cases du damier détectés dans l'image numérique (points de coordonnées (x,y)). Pour cela, il faut donc définir l'opération inverse de l'opération de projection vue en cours:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \rightarrow \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} \rightarrow \begin{bmatrix} x = x'/z' \\ y = y'/z' \\ z = z'/z' = 1 \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \end{bmatrix}$$

où le passage (X,Y,Z) à (x',y',z') se fait par la matrice K.

Déterminer, sur feuille, les coordonnées (X,Y,Z) en fonction de (x,y), en faisant apparaître la matrice inverse de la matrice K, que l'on notera K^{-1} , et dont on donnera l'expression ainsi que les coefficients. On vérifiera que ce résultat (i.e. K^{-1}) est conforme au résultat fourni par numpy.linalg.inv.

2.2 Détection automatique des coordonnées 2D « pixels » des coins des cases du damier

Ecrire le script Python permettant la détection automatique des coins du damier dans l'image numérique de référence fournie (fonction opencv « findChessboardCorners »). On affichera l'image sur laquelle on aura superposé les coins détectés (fonction opencv « drawChessboardCorners », à invoquer avant « imshow » – voir figure 1).

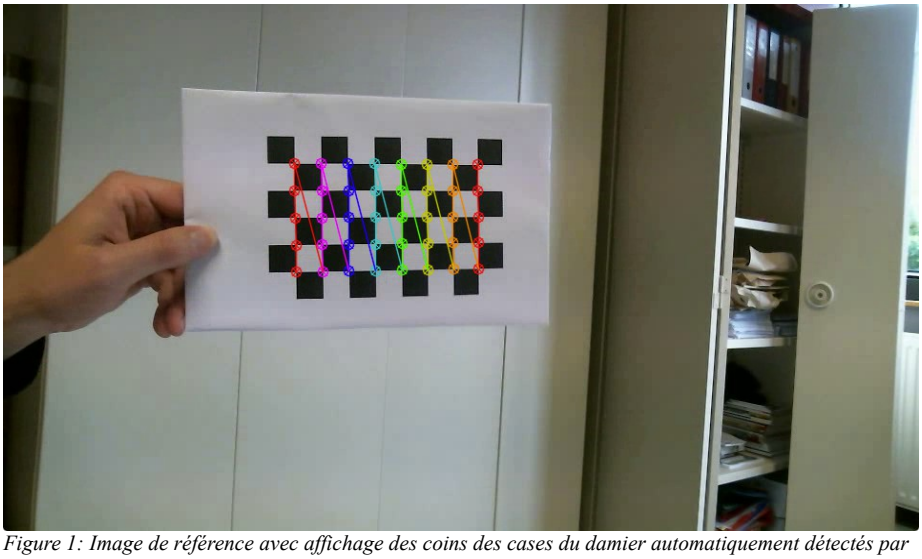

Figure 1: Image de référence avec affichage des coins des cases du damier automatiquement détectés par opencv

On donne un caneva de départ pour lire et afficher une image avec OpenCV

```
import numpy as np ;import cv2

image = cv2.imread('reference_image.png') #image est de type numpy.array
cv2.imshow('frame',image)
cv2.waitKey()
cv2.destroyAllWindows()
```

Lecture et affichage d'une image avec OpenCV (« image » étant un tableau « numpy »)

Indication: La matrice des coordonnées des points détectés retournée par « findChessboardCorners » est mal conditionnée : il est nécessaire de la « reformer » de manière à avoir n lignes (autant de lignes que de coins) et 2

colonnes (coordonnées x et y). Pour cela, on utilisera la méthode « .reshape(-1,2) ».

Les coordonnées des points détectés que vous devez trouver sont :

```
[[ 497.04840088  164.49719238]
 [ 497.34249878  191.22937012]
 [ 497.53610229  217.64170837]
 [ 497.61395264  244.35397339]
 [ 497.95266724  270.93353271]
...
 [ 333.99298096  218.56646729]
 [ 334.41964722  246.19784546]
 [ 334.62289429  273.60757446]
 [ 305.05355835  163.07861328]
 [ 305.23812866  190.86212158]
 [ 306.01153564  218.54602051]
 [ 306.51489258  246.5005188 ]
 [ 306.61694336  274.18325806]]
```

2.3 Programmation des deux fonctions « coordonnées 2D pixels ↔ coordonnées 3D caméra »

La première fonction est dédiée au passage « coordonnées pixels 2D → coordonnées 3D caméra ». Elle aura pour prototype : « pixel2camera(K,Z,points2D) » , où K est la matrice de calibration, Z la côte des points 2D, et points2D désigne la liste des points 2D (tableau 2D : un point par ligne ; x et y étant les deux colonnes). Cette fonction retourne les points 3D (tableau 2D : un point par ligne ; x, y et z étant les trois colonnes).

Pour tester cette première fonction, on utilisera le coordonnées 2D des coins du damier, K et Z étant connus (Z=500). En interne à cette fonction, on calculera automatiquement la matrice inverse K^{-1} . Pour l'image de référence et la matrice K considérées, on doit obtenir la liste de points 3D suivante :

```
[[  10.87449439  -65.16349882  500.        ]
 [  11.0620877   -48.65244787  500.        ]
 [  11.18557965  -32.33894483  500.        ]
 [  11.23523728  -15.84019291  500.        ]
 [  11.4512898     0.57659374  500.        ]
 …
 [-110.98029619  -31.7803991   500.        ]
 [-110.6592249   -14.51438545  500.        ]
 [-110.59413078    2.58377703  500.        ]]
```

La seconde fonction est dédiée au passage « coordonnées 3D caméra → coordonnées pixels 2D ». Elle aura pour prototype : world2pixel(K,points3D), où K est la matrice de calibration et « points3D » désigne la liste des points 3D (tableau 2D : un point par ligne ; x, y et z étant les trois colonnes). Cette fonction retourne les points 2D (tableau 2D : un point par ligne ; x et y étant les deux colonnes).

Pour tester cette seconde fonction, on utilisera les coordonnées 3D des coins du damier calculés par la première fonction, et on affichera les points 2D obtenus (fonction opencv « drawChessboardCorners »).
On pourra également vérifier les valeurs numériques :

```
[[ 497.04840088  164.49719238]
 [ 497.34249878  191.22937012]
 [ 497.53610229  217.64170837]
 ...
 [ 305.23812866  190.86212158]
 [ 306.01153564  218.54602051]
 [ 306.51489258  246.5005188 ]
 [ 306.61694336  274.18325806]]
```

On conservera les deux fonctions obtenues dans le fichier « helper.py » pour réutilisation ultérieure.

Indication 1 : Pour le produit matriciel (i.e. Entre deux matrices ou une matrice et un vecteur), on utilisera la fonction « numpy.dot ». Pour un produit de type « C=numpy.dot(A,B) », attention à ce que B soit « bien orientée » afin que le nombre de lignes de B soit égal au nombre de colonne de A. Si ce n'est pas le cas, pour pourra au préalable transposer B, et re-transposer le résultat : par exemple « C=numpy.dot(A,B.T).T », où la méthode « .T » retourne le tableau (ou plutôt matrice) transposé(e).
Indication 2 : On peut ajouter une colonne (par exemple de « 1 ») à un tableau en utilisant numpy.hstack((tab,col)), avec col=numpy.ones((nb_lignes,1)), où nb_lignes=tab.shape[0].