BLAIN Hugo ESIREM 4A IE

Compte Rendu Analyse d'Image

TP1 Reconstruction 3D

Table des matières

Introduction :

L'objectif de ce TP est de se familiariser avec la reconstruction 3D à travers toutes ces étapes, de l'acquisition aux résultats, du calibrage à la carte de profondeur. Nous aurons donc une partie acquisition, puis une partie traitement d'images. Nous effectuerons la calibration d'une caméra à partir d'un échantillon d'image d'un échiquier et nous analyserons les résultats. Pour ceci, nous travaillerons à partir d'un script python, « Modélisation.py », fournis avec l'énoncé du TP et utiliserons la librairies openCV pour la première fois.

Avant de commencer je tiens à m'excuser, l'appréhension de openCV a été plus longue de prévu, je n'ai donc pas pu aller plus loin que l'exercice 1. Aussi, après plusieurs tentatives qui furent des échecs avec des images personnels, j'ai finis par utiliser celles fournies.

Exercice 1 : Calibrage de la caméra

Introduction :

Le but de cette exercice est d'à partir d'un échantillon de photo d'un échiquier, déterminer la matrice de calibration de la caméra du téléphone portable.

1.1 Acquisition

Pour réaliser le calibrage de cette caméra nous utiliserons les images qui nous ont été fournies, contenues dans le dossier « Images/chess/P30 ».

Figures 1 et 2 : Exemples d'images de notre échantillons

1.2 Estimation de la matrice de la caméra

Pour déterminer la matrice de calibrage de la caméra nous nous intéresserons à la fonction CameraCalibration() de notre script « Modélisation.py ». Nous commençons par adapter et commenter le code fournis, voir anexe.

Pour fonctionner nous devons dans un premier temps, définir les dimensions de notre échiquier et déclarer deux tableaux où seront stockés les coordonnées des coins détectés, en 3D et 2D dans le plan de l'image. Pour toute la suite, on définit comme « coins » dans l'échiquier, toutes les coins des carreaux, c'est-à-dire l'intersections noir/blanc entre 4 cases par exemple.

Ensuite, pour chaque image de notre échantillon, les coins sont détectés grâce à la méthode « cv.findChessboardCorners() » et renvoie true. Cependant il est possible que la détection échoue pour certaines images, dans ce cas la fonction renverra false. A noter que je me suis permis de compléter l'affichage afin de mieux comprendre à quoi correspondait les sorties dans le terminal.

```
C:\Users\yotex\AppData\Local\Programs\Python\Python39\python.exe C:/
Résulat détection des coins pour l'image n° 0 :  True
Résulat détection des coins pour l'image n° 1 :  True
Résulat détection des coins pour l'image n° 2 :  True
Résulat détection des coins pour l'image n° 3 :  True
Résulat détection des coins pour l'image n° 4 :  True
Résulat détection des coins pour l'image n° 5 :  False
Résulat détection des coins pour l'image n° 6 :  True
Résulat détection des coins pour l'image n° 7 :  False
Résulat détection des coins pour l'image n° 8 :  True
Résulat détection des coins pour l'image n° 9 :  True
Résulat détection des coins pour l'image n° 10 :  True
Résulat détection des coins pour l'image n° 11 :  True
Résulat détection des coins pour l'image n° 12 :  False
Résulat détection des coins pour l'image n° 13 :  False
Résulat détection des coins pour l'image n° 14 :  True
Résulat détection des coins pour l'image n° 15 :  False
Résulat détection des coins pour l'image n° 16 :  True
Résulat détection des coins pour l'image n° 17 :  True
Résulat détection des coins pour l'image n° 18 :  True
```

Figure 3 : Détection des coins sur les images avec « cv.findChessboardCorners() »

Une fois les coins détectés pour une image, on affine leurs positions avec la fonction « cv.cornerSubPix() » et on les dessine sur l'image avec la méthode « cv.drawChessboardCorners() ». Cela nous permet de vérifier visuellement si la détection a été correctement effectuée.

Figure 4 : Exemple d'affichage après détection, affinage et dessin des coins

Enfin on fait appelle à la fonction « cv.calibrateCamera() » afin d'obtenir les paramètres intrinsèques et extrinsèques de la caméra. On récupèrera entre autres la matrice de calibrage recherchée qu'on déclare « mtx » et les coefficients de distorsions, « dist ».

```
Résultats de cv.calibrateCamera:
camraMatrix
 [[1.41648092e+03 0.00000000e+00 6.67598677e+02]
 [0.00000000e+00 1.41360922e+03 9.16650426e+02]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]

dist
 [[-0.00996315  0.66100357  0.00470359 -0.00439002 -1.82246322]]
```

Figure 5 : Résultats de la fonction « cv.calibrateCamera() »

1.3 Analyse des résultats

Une partie du code permet d'enlever la distorsion induite par l'appareil photo. En effet avec la plupart des caméra de téléphone portable on observe un effet « fish-eye » qui grossit la zone centrale de la photo.

Pour corriger cette effet, nous calculons une nouvelle matrice de calibrage pour la caméra en optimisant l'ancienne à l'aide de la fonction « cv.getOptimalNewCameraMatrix() ».

```
Résultats de cv.calibrateCamera:
camraMatrix
 [[1.41648092e+03 0.00000000e+00 6.67598677e+02]
 [0.00000000e+00 1.41360922e+03 9.16650426e+02]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]

dist
 [[-0.00996315  0.66100357  0.00470359 -0.00439002 -1.82246322]]

newcameramtx
 [[1.00423450e+03 0.00000000e+00 5.52874103e+02]
 [0.00000000e+00 7.90380249e+02 1.13082223e+03]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]
```

Figure 6 : Nouvelle matrice de calibrage

Et ensuite on effectue la correction grâce à la méthode « cv.initUndistortRectifyMap() » et on stocke l'image corrigée sous le nom « calibresultM.png ».

Figure 7 et 8 : Exemple de correction de l'effet fish-eye sur l'image « IMG_20201206_093855.jpg ». A gauche l'image d'origine et à droite l'image corrigée « calibresultM.png ».

On peut remarquer deux choses, d'abord on peut dire que l'effet fish-eye n'est pas très marqué sur l'image d'origine. Et deuxièmes, la correction a engendrer une déformation, voir coin inférieur droit de la figure 8, la grille s'entend vers l'extérieur. L'effet fish-eye a bien disparu mais la grille est déformée.

Je soupçonne une erreur/imprécision lors de la reprojection des points avec la méthode « cv.remap « . Peut-être que les arguments que nous lui passons en paramètre ne sont pas suffisamment précis. En effet la suite du code nous permet de calculer la valeur de l'erreur de reprojection, 0 indiquerait que notre reprojection est parfaite mais nous obtenons 0. 4371897309610618.

```
total error: 0.4371897309610618
```

Figure 9 : Valeur globale de l'erreur de reprojection

Conclusion

Dans ce TP nous aurons découvert openCV et manipuler des notions fondamentales de l'analyse d'image vues en cours. Nous aurons réussis à obtenir la matrice de calibrage de la caméra du téléphone. Et pour finir, nous aurons tenté de corriger l'effert fish-eye mais malheureusement une déformation a lieu au cours de cette correction.

Annexe

Dépôt Github : https://github.com/HugoBlain/TP1_Analyse_dimage

```
# Nombre de coins intérieurs par ligne et colonne d'échiquier
# Pour trouver ca, compter le nombre d'intercection noir/blanc sur
largeur/longueur de l'échiquier -> ici 4 et 6
largeur = 4
longueur = 6
# largeur des carreaux du damier en mm
largeurCarreaux = 40


import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
from scipy import ndimage
import glob
```

```
# Exercice 1
def CameraCalibration():
    # critères d'arret --> epsilon = 0.001 , max iteration = 30
    criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 30,
0.001)

    # prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)
    objp = np.zeros((largeur * longueur, 3), np.float32)
    objp[:, :2] = np.mgrid[0:largeur, 0:longueur].T.reshape(-1, 2)
    # Mise à l'echelle
    objp[:, :2] *= largeurCarreaux

    # tableaux pour stocker les points pour toutes les images de
l'échiquier
    objpoints = []  # points en 3D
    imgpoints = []  # points en 2D dans le plan image

    # forme une seule variable pour toutes les images, cela nous permettra
de toutes les parcourir plus facilement
    # liste de nom sous forme de chaine
    images = glob.glob('./Images/chess/P30/*.jpg')

    # pour chaque image (voir ligne juste au dessus pour images)
    for i, fname in enumerate(images):
        # on lit l'image que l'on traitera lors de cette itération avec
cv.imread()
        img = cv.imread(fname)
        # on brouille l'image et la sous-échantillonne avec cv.pyrDown()
        img = cv.pyrDown(img)
        # Convertit l'image d'un espace colorimétrique à un autre avec
cv.cvtColor
        # arguments: InputArray -> img: l'image traitée lors de cette
itération
        #            Code       -> cv.COLOR_BGR2GRAY: code pour la
conversion de RVB/BGR en niveaux de gris
        gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

        # Trouver les coins de l'échiquier
        # arguments : InputArray  -> gray : Vue de l'échiquier source. Il
doit s'agir d'une image en niveaux de gris ou en couleurs 8 bits.
        #             Size        -> (largeur, longueur) : Nombre de coins
intérieurs par ligne et colonne d'échiquier
        #             OutputArray -> None : ici on récupère direct les
coins detecté dans la variable corners
        #             ret prend la valeur true si les coins sont trouvés et
false sinon
        ret, corners = cv.findChessboardCorners(gray, (largeur, longueur),
None)

        # on affiche le résultat de findChessboardCorners
        print("Résulat détection des coins pour l'image n°", i,": ", ret)
```

```
# Si suffisamment de coins ont été trouvé
    if ret == True:
        # Ajoute à la liste des points des objets une copie des points des
objets préparés lignes 26/27
        objpoints.append(objp)
        # Affine les emplacements des coins
        # arguments : InputArray -> gray: image source, en 8 bits
        #             Corners    -> corners: Coordonnées initiales des
coins d'entrée
        #             WinSize    -> (11, 11): La moitié de la longueur du
côté de la fenêtre de recherche. Par exemple, si winSize = Size (5,5),
alors un ( 5 * 2 + 1 ) × ( 5 * 2 + 1 ) = 11 × 11
        #             ZeroZone   -> (-1, -1): La moitié de la taille de la
région morte au milieu de la zone de recherche sur laquelle la sommation
dans la formule ci-dessous n'est pas effectuée.
        #                                   Il est parfois utilisé pour
éviter d'éventuelles singularités de la matrice d'autocorrélation.
        #                                   La valeur de (-1, -1) indique
qu'une telle taille n'existe pas.
        #             Criteria   -> criteria: Critères de fin du processus
itératif de raffinement d'angle.
        #                                   Autrement dit, le processus de
raffinement de la position d'angle s'arrête soit après les itérations de
critère.maxCount, soit lorsque la position d'angle se déplace de moins que
critère.epsilon sur une itération.
        #             Sortie     -> corners2: coordonnées raffinées des
coins
        corners2 = cv.cornerSubPix(gray, corners, (11, 11), (-1, -1),
criteria)
        # on ajoute à la liste des points des images, la position des coins
trouvés
        imgpoints.append(corners)
        # Dessine sur l'échiqiuer les coins qui ont été détectés. Soit
comme des cercles rouges si il n'a pas été trouvé, soit comme des points
colorés reliés par des lignes si il a été trouvé
        # arguments: InputOutputArray -> img: l'image de l'itération cours
sans la conversion en niveaux de gris, sur laquelle on va dessiner
        #            PatternSize      -> (largeur, longueur): Nombre de
coins intérieurs par ligne et colonne d'échiquier
        #            Corner           -> corners2: coordonnées raffinées
des coins
        #            PatternWasFound  -> ret: retour de
findChessboardCorners, ici forcement true grâce au "if" ligne 61, true =
coins detectés
        cv.drawChessboardCorners(img, (largeur, longueur), corners2, ret)
        # affichage de l'image sur laquelle on vient de dessiner dans une
fenetre
        cv.namedWindow('img', 0)
        cv.imshow('img', img)
        # permet de faire une pause
        #    waitKey attend qu'on presse une touche. Ici la fonction attend
pendant seulement 0.5s comme on lui a indiqué passant un argument (500 ms)
        cv.waitKey(500)
# on ferme toutes les fenetres
cv.destroyAllWindows()
```

```
# Recherche les paramètres intrinsèques et extrinsèques de la caméra
# arguments: InputArrayOfArrays -> objpoints: tableau des points en 3D
connus
#            InputArrayOfArrays -> imgpoints: tableau des coordonnées 2D
des coins connus
#            Size               -> gray.shape[::-1]: taille de l'image (ici
1368 x 1824)
#            Flags              -> None
#            Criteria           -> None
# sorties: boolean     -> ret: valeur de retour, true ou false si c'est une
résussite ou un échec
#          OutputArray -> mtx: Matrice intrinsèque de caméra (Matrice de
calibrage 3x3)
#          OutputArray -> dist: Coefficients de distorsion de l'objectif
#          OutputArray -> rvecs: vecteurs de rotation estimé pour chaque
vue
#          OutputArray -> tvecs: vecteurs de traduction estimés pour chaque
vue
ret, mtx, dist, rvecs, tvecs = cv.calibrateCamera(objpoints, imgpoints,
gray.shape[::-1], None, None)
print("\nRésultats de cv.calibrateCamera:")
print('camraMatrix\n', mtx, "\n")
print('dist\n', dist, "\n"  )
```

```
# on selectionne une nouvelle image (imread) et on la sous-echantillonne
(pyDown)
img = cv.pyrDown(cv.imread('./Images/chess/P30/IMG_20201206_093855.jpg'))
# Dimension de l'image -> h = hauteur, w = largeur
h, w = img.shape[:2]
# Calcule d'une nouvelle matrice de calibrage en optimisant celle
précedement trouvée
# Ici ceci nous permet de corriger la distorsion induite par l'appareil
photo
# arguments: InputArray           -> mtx: matrice de calibrage d'entrée
#            InputArray           -> dist: tableau des coefficients de
distorsion
#            ImageSize            -> (w, h): taille de l'image d'origine
#            Alpha                -> 1: Paramètre de mise à l'échelle libre
entre 0 (lorsque tous les pixels de l'image non déformée sont valides) et 1
(lorsque tous les pixels de l'image source sont conservés dans l'image non
déformée).
#            newImgSize           -> (w, h): taille de l'image après
rectification
#            centerPrincipalPoint -> None: Indicateur facultatif qui
indique si, dans la nouvelle matrice intrinsèque de la caméra, le point
principal doit être au centre de l'image ou non.
#                                    Par défaut, le point principal est
choisi pour adapter au mieux un sous-ensemble de l'image source (déterminé
par alpha) à l'image corrigée.
# sortie: newcameramtx: Nouvelle matrice de calibrage obtnue apreès la
rectification
#         roi: Rectangle de sortie facultatif qui décrit la région de tous
les bons pixels dans l'image non déformée.
newcameramtx, roi = cv.getOptimalNewCameraMatrix(mtx, dist, (w, h), 1, (w,
h))
# on affiche la nouvelle matrice
print('newcameramtx\n', newcameramtx)
```

```
# On souhaite corriger la distortion
#  Calcule la carte de transformation
# arguments: InputArray -> mtx: matrice de calibrage de la caméra avant
rectification
#            InputArray -> dist: tableau des coefficients de distorsion
#            InputArray -> None: Transformation de rectification
optionnelle dans l'espace objet (matrice 3x3)
#            InputArray -> newcameramtx: matrice de calibrage apres
rectification
#            ImageSize  -> (w, h): taille de l'image non déformée
#            M1Type     -> 5: type de la première carte de sortie
mapx, mapy = cv.initUndistortRectifyMap(mtx, dist, None, newcameramtx, (w,
h), 5)
# utilisation d'une fonction de remappage pour cooriger la distorion
dst = cv.remap(img, mapx, mapy, cv.INTER_LINEAR)
# Enfin on recadre l'image
x, y, w, h = roi
dst = dst[y:y + h, x:x + w]
# On affiche le résultat
cv.namedWindow('img', 0)
cv.imshow('img', dst)
# 0 en paramètre de waitKey signifit qu'il attendra pour toujours tant
qu'on ne presse pas de touche
cv.waitKey(0)
# on stocke le résultat en créant une nouvelle image
cv.imwrite('./Images/Resultats/calibresultM.png', dst)

# variable pour l'erreur de re-projection
# Donne une bonne estimation de l'exactitude des paramètres trouvés. Plus
l'erreur de re-projection est proche de zéro, plus les paramètres que nous
avons trouvés sont précis
mean_error = 0
for i in range(len(objpoints)):
    #  nous devons d'abord transformer l'objet point en point image en
utilisant cv.projectPoints ()
    imgpoints2, _ = cv.projectPoints(objpoints[i], rvecs[i], tvecs[i], mtx,
dist)
    # Ensuite, nous pouvons calculer la norme absolue entre ce que nous
avons obtenu avec notre transformation et l'algorithme de recherche de coin
    error = cv.norm(imgpoints[i], imgpoints2, cv.NORM_L2) / len(imgpoints2)
    mean_error += error
# Pour trouver l'erreur moyenne, nous calculons la moyenne arithmétique des
erreurs calculées pour toutes les images d'étalonnage
# et on affiche
print("\ntotal error: {}".format(mean_error / len(objpoints)))

# on retourne la matrice de calibrage de notre caméra
return newcameramtx
```