Bachelier Baptiste

R4.22 :
Projet scanner 3D stéréo

Objectif :

Pour ce projet on doit mettre en place un scanner 3D stéréo utilisant 2 caméras DMK 33GX174 de chez The Imaging Source et Python pour traiter les images.

Démarche :

Pour réaliser ce projet, on va suivre les étapes suivantes :

I/ Acquisition des images
II/ Calibration mono des caméras
III/ Calibration stéréo
IV/ Recherche de points d'intérêt dans les images
V/ Triangulation des points d'intérêts
VI/ Affichage de la représentation 3D de l'objet

Les différentes parties seront présentées dans l'ordre donné si-dessus. Le code du projet est disponible sur le git : https://github.com/Ssea2/scanner_3D_stereo

I/ acquisition des images

Dans un premier temps j'ai cherché la documentation des caméras (1), les caméras ne communiquent pas en http ou rtsp mais utilisent GigE Vision/IC Imaging Control. Pour communiquer avec les caméras, Mr.Druon a parlé de Aravis, cette librairie existe en Python donc j'ai essayé de l'utiliser pour voir le flux vidéo des caméras. Pour installer et utiliser cette librairie il y a eu quelques problèmes :

- Les exemples de code que je faisais générer par chatgpt ne marchaient pas, j'ai fini par comprendre que c'était une différence de version entre la doc (2) que j'ai donnée à chatgpt/claude (v0.8) et la version que j'ai installée (v0.0.1).

- J'ai testé une alternative avec gi et non pip, mais ça n'a pas marché car bien que « import gi » était marqué comme installé il ne l'était pas, donc il n'arrivait pas à trouver Aravis.

Ensuite, comme le projet peut se faire à plusieurs donc j'ai laissé mes camarades faire le code pour récupérer les images pour pouvoir avancer avec la calibration de la caméra. Leur code utilise les librairies « Tcam, v1.0 », « Gst, v1.0 » et « Glib, v2.0 » pour initialiser une variable caméra qui va pouvoir récupérer les images et les paramètres de la caméra dont on met le numéro de série en paramètres. Avec cette variable on choisit où on doit sauvegarder les fichiers puis on lance l'acquisition d'image. On attend 2 seconde avant de les enregistrer dans file_location_1 et file_location_2.

```python
def capture_images():
    serial_1 = "32910328"
    serial_2 = "32910323"

    pipeline_1 = Gst.parse_launch("tcambin name=bin1"
                                  " ! video/x-raw,format=BGRx,width=1920,height=1080,framerate=30/1"
                                  " ! videoconvert"
                                  " ! jpegenc"
                                  " ! filesink name=fsink1")

    pipeline_2 = Gst.parse_launch("tcambin name=bin2"
                                  " ! video/x-raw,format=BGRx,width=1920,height=1080,framerate=30/1"
                                  " ! videoconvert"
                                  " ! jpegenc"
                                  " ! filesink name=fsink2")

    if serial_1 is not None:
        camera_1 = pipeline_1.get_by_name("bin1")
        camera_1.set_property("serial", serial_1)

    if serial_2 is not None:
        camera_2 = pipeline_2.get_by_name("bin2")
        camera_2.set_property("serial", serial_2)

    timestamp = time.strftime("%Y%m%d_%H%M%S")
    #On définit le path pour enregistrer les images
    file_location_1 = f"camera1_3D/camera_1_image_{timestamp}.jpg"
    file_location_2 = f"camera2_3D/camera_2_image_{timestamp}.jpg"

    fsink_1 = pipeline_1.get_by_name("fsink1")
    fsink_1.set_property("location", file_location_1)

    fsink_2 = pipeline_2.get_by_name("fsink2")
    fsink_2.set_property("location", file_location_2)

    pipeline_1.set_state(Gst.State.PLAYING)
    pipeline_2.set_state(Gst.State.PLAYING)

    print(f"Capturing images... ({file_location_1}, {file_location_2})")
    time.sleep(2)
    pipeline_1.set_state(Gst.State.NULL)
    pipeline_2.set_state(Gst.State.NULL)
    print("Images saved!")
```

Code 1 : issue du repo github de Mathéo (« linux only/get_picture.py »)

Pour la connexion aux caméras, on a aussi eu des problèmes avec leurs adresses IP. Lors de la première séance, Mehdi a utilisé le logiciel GigECam IP Config pour trouver les caméras est leurs assigner une IP en 10.216.x.y/16.

Cependant, pendant les heures de projet il arrive que les caméras changent d'IP pour une adresse APIPA ou en IP privée/24 et on ne sait pas pourquoi. Il faut donc relancer le logiciel remettre une IP dans la bonne plage et continuer.

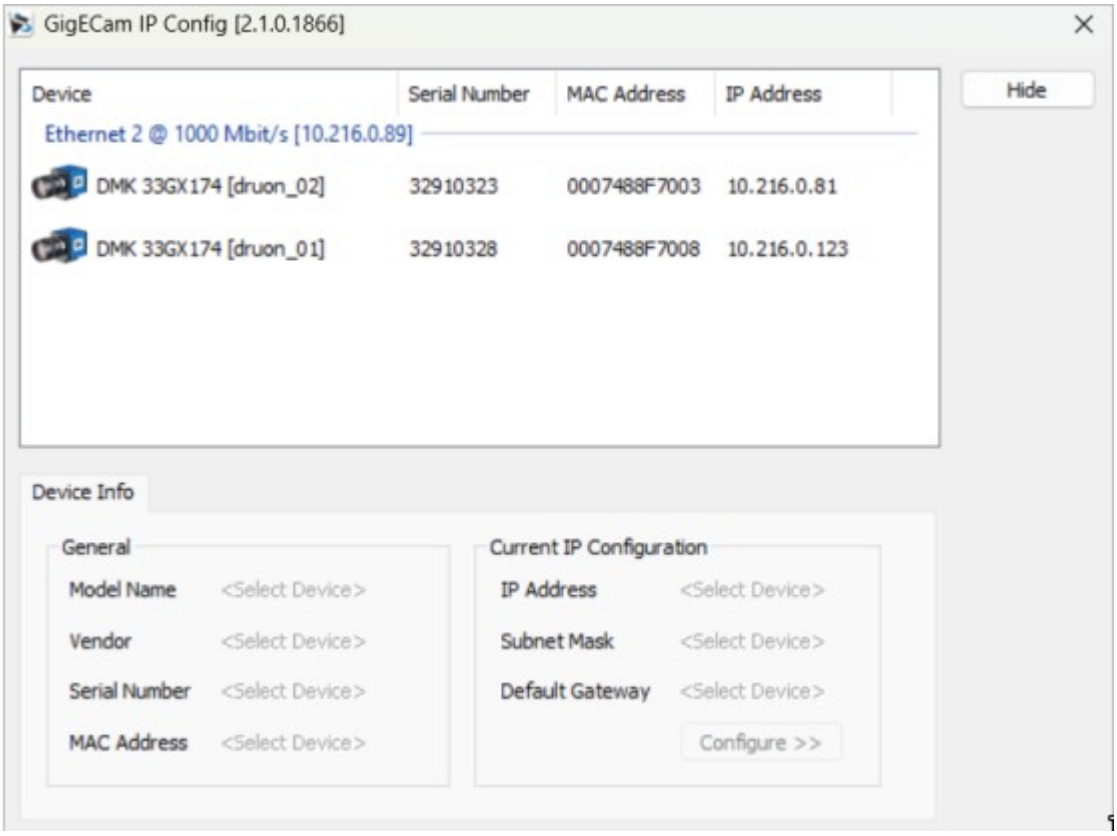

Fig 1 : screen du panel de config de GigECam, screen issue du compte rendu d'Aurélien

II/ calibration mono

Pour cette partie j'ai commencé par récupérer des images contenant un chessboard en utilisant une webcam. Le code utilisé est le suivant (Code 2) :

```
def get_image():
    url = "/dev/video2"

    cap = cv.VideoCapture(url)
    c=0
    while True:
        ret, frame = cap.read()

        if ret == True:
            cv.imshow("t", frame)
            cv.waitKey(0)
            cv.destroyAllWindows()
            cv.imwrite(f"img_{c}.jpg",frame)
            c+=1
        else:
            print("pas d'img")
            break
```

Code 2 : recupération es images d'un webcam

La caméra est connectée en USB à mon PC et les données arrivent via le « port » /dev/video2, dès que j'appuie sur une touche la caméra enregistre une image. Ce code a pour avantage d'être rapide, je n'ai pas à faire clic droit enregistrer etc mais il donne beaucoup d'images qu'il faut ensuite trier. On observe dans le screen ci-dessous qu'il y a eu 180 images de prise et j'en utilise qu'une dizaine.

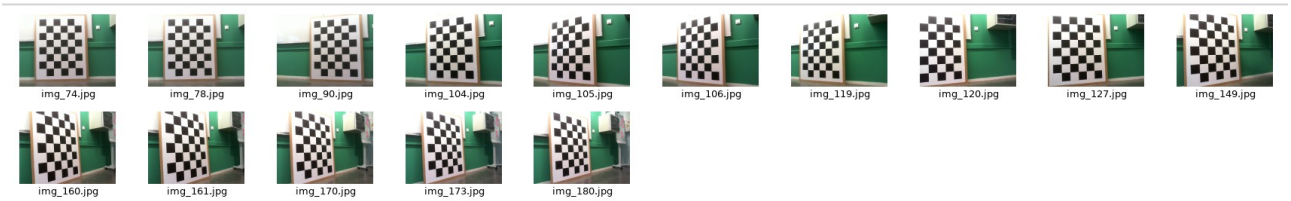

Fig 2 : Screenshot montrant les images de test pour la calibration mono

Pour réaliser la calibration j'ai utilisé la documentation d'OpenCV (3) pour créer cette fonction (c'est un copier-coller de la doc mais adapté à notre mire de calibration). Dans le code ci-dessous (code 3) j'ai modifié X et Y pour qu'elles correspondent aux dimensions de la mire-1, pour les images ci-dessus (Fig 2) X=5 et Y=7. Cette fonction nous renvoie les paramètres intrinsèques de la caméra (la calibration mono est bonne quand les valeurs de Cx et Cy correspondent aux dimentions de l'image divisée par 2 si on calcule les parametres en pixel) et la liste des coefficients de distorsion.

$$camera\ matrix = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Fig 3 : matrice des paramètres intrasèque

```python
def calcam(img_folder, chessboad_size, square_size):
    """
    @param:
        -img_folder :str: chemin vers le dossier qui contient les images de calibration
    @return:
        -mtx :array: la matrice des parametres intrinsèque de la caméra
        -dist :array: la list des coeficients de distortion
    """
        # termination criteria
    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((chessboad_size[0]*chessboad_size[1],3), np.float32)
    objp[:,:2] = np.mgrid[0:chessboad_size[1],0:chessboad_size[0]].T.reshape(-1,2) * square_size

    # Arrays to store object points and image points from all the images.
    objpoints = [] # 3d point in real world space
    imgpoints = [] # 2d points in image plane.
    img_size = None

    images = glob.glob(img_folder)
    #print(len(images))

    for fname in images:
        img = cv.imread(fname)
        if img is None:
            print(f"Warning: Could not read image {fname}")
            continue
        gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
        img_size=gray.shape[::-1]
        # Find the chess board corners
        ret, corners = cv.findChessboardCorners(gray, (chessboad_size[1],chessboad_size[0]), None)

        # If found, add object points, image points (after refining them)
        if ret == True:
            #print(ret, fname, img.shape)
            objpoints.append(objp)

            corners2 = cv.cornerSubPix(gray,corners, (11,11), (-1,-1), criteria)
            imgpoints.append(corners2)
            #print("\n\n",corners2)
            # Draw and display the corners
            cv.drawChessboardCorners(img, (chessboad_size[1],chessboad_size[0]), corners2, ret)
            cv.imshow('img', img)
            cv.waitKey(500)


    cv.destroyAllWindows()
    ### A TESTER transformer fx fy en mm
    """print(gray.shape)
    print("mtx\n",mtx)
    print("dist\n",dist)
    print("rvecs\n",rvecs)
    print("tvecs\n",tvecs)"""
    ret, mtx, dist, rvecs, tvecs = cv.calibrateCamera(objpoints, imgpoints, img_size, None, None)
    return mtx,dist
```

Code 3 : calibration d'une caméra

Pour la caméra 1, j’ai la matrice suivante :

Mx1=[[2.82550979e+03, 0.00000000e+00, 1.00859885e+03]
 [0.00000000e+00, 2.82912963e+03, 6.69132111e+02]
 [0.00000000e+00, 0.00000000e+00, 1.00000000e+00]]

et pour la caméra 2 j’ai :

Mx2=[[2.88615917e+03, 0.00000000e+00, 1.04558730e+03]
 [0.00000000e+00, 2.88584584e+03, 6.25220618e+02]
 [0.00000000e+00, 0.00000000e+00, 1.00000000e+00]]

De ce que j’ai compris de cette étape il se sert de la différence de linéarité entre les points qu’il trouve dans l’image qui forme des arcs de cercle et la matrice de coin qui est un damier droit pour calculer la déformation et en déduire les paramètres intrasèque et extrasèque de la caméra.

Maintenant que les 2 caméras sont calibrées indépendamment il faut les calibrer ensemble afin d’avoir les transformations pour passer de l’un à l’autre.

III/ calibration stéréo

Pour la calibration stéréo, j’ai réutilisé la fonction « calcam » que j’ai modifié pour exécuter la calibration, comme exemple j’ai la fonction de calibration stéréo d’un git (4). La fonction consiste à récupérer les coins detectés dans les 2 images et à en déduire les matrices de transformation pour passer de l’un à l’autre.

```python
def calstereo(mtx1,mtx2,dist1,dist2, folder_cam_left, folder_cam_right, chessboad_size, square_size):
    stereocalibration_flags = cv.CALIB_FIX_INTRINSIC
    criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 30, 0.001)


    objp = np.zeros((chessboad_size[0]*chessboad_size[1],3), np.float32)
    objp[:,:2] = np.mgrid[0:chessboad_size[1],0:chessboad_size[0]].T.reshape(-1,2) *square_size

    # Arrays to store object points and image points from all the images.
    objpoints = [] # 3d point in real world space
    imgpoints_left = [] # 2d points in image plane.
    imgpoints_right= []
    #img_size = None

    images_left = sorted(glob.glob(folder_cam_left))
    #print(images_left[0])

    images_right = sorted(glob.glob(folder_cam_right))
    #print(images_right[0])
    for Lfname, Rfname in zip(images_left,images_right):
        # cam1
        Limg = cv.imread(Lfname)

        #cam2
        Rimg = cv.imread(Rfname)
        if Rimg is None or Limg is None:
            print(f"Warning: Could not read images {Lfname} or {Rfname}")
            continue

        Lgray = cv.cvtColor(Limg, cv.COLOR_BGR2GRAY)
        Rgray = cv.cvtColor(Rimg, cv.COLOR_BGR2GRAY)
        img_size=Lgray.shape[::-1]


        # Find the chess board corners
        ret, Lcorners = cv.findChessboardCorners(Lgray, (chessboad_size[1],chessboad_size[0]), None)
        ret, Rcorners = cv.findChessboardCorners(Rgray, (chessboad_size[1],chessboad_size[0]), None)

        if Rcorners is None or Lcorners is None:
            print(f"Warning: pas trouver le damier images {Lfname} or {Rfname}")
            continue

        if Lcorners.all() == None:
            print("L: ",None)
        if Rcorners.all() == None:
            print("R: ",None)

        # If found, add object points, image points (after refining them)
        if ret == True:
            objpoints.append(objp)

            corners2 = cv.cornerSubPix(Lgray,Lcorners, (11,11), (-1,-1), criteria)
            imgpoints_left.append(corners2)

            corners1 = cv.cornerSubPix(Rgray,Rcorners, (11,11), (-1,-1), criteria)
            imgpoints_right.append(corners1)

            #print("\n\n",corners2)
            # Draw and display the corners
            cv.drawChessboardCorners(Limg, (chessboad_size[1],chessboad_size[0]), corners2, ret)
            cv.imshow('Left', Limg)
            cv.waitKey(500)

            cv.drawChessboardCorners(Rimg, (chessboad_size[1],chessboad_size[0]), corners1, ret)
            cv.imshow('Right', Rimg)
            cv.waitKey(500)

        cv.destroyAllWindows()

    ret, CM1, dist1, CM2, dist2, R, T, E, F = cv.stereoCalibrate(objpoints, imgpoints_left, imgpoints_right, mtx1, dist1,
    mtx2, dist2, img_size, criteria = criteria, flags = stereocalibration_flags)

    return R,T
```

Code 4 : calibration stéréo basé sur l fonction calcam

Dans cette fonction on fait * « sqarre_size » à la variable « objp [:,:2] » afin d'avoir les valeurs en cm/mm. Dans mon cas j'ai choisi de mettre les valeurs en mm, donc « squarre_size » vaut 160mm. C'est en regardant le code de Mathéo que j'ai vue ça et que j'ai compris pourquoi il l'avait mis, avec les valeurs ont plus de sens.

Fig 4: Images de la caméra 2 pris par mes camarades

Avec les images prises par mes camarades j'obtiens ces matrices:

stéréo Rotation :
[[0.99981046, 0.00485584, -0.01885369],
[-0.00452883, 0.99983924, 0.01734903],
[0.0189349, -0.01726036, 0.99967172]]

stéréo translation: [[-305.9072207],[1.26129133], [92.84330504]]

On observe que le vecteur de translation contient un déplacement en x de -305.9 mm, or je sais que les caméras sont à environ 30cm de distance donc la calibration stéréo est plutôt bonne.

Une fois que la calibration stéréo est faite il faut récupérer les points d'intérêts pour pouvoir utiliser ces matrices pour la reconstruction 3D.

IV/ recherche de point d'interet dans les images

Pour cette partie j'ai choisi d'utiliser Sift pour trouver le point d'intérêt dans une image, j'ai fait 2 programmes. Le premier (code 5) c'est un mixte de ce que m'a donné Claude.ia, de la documentation d'OpenCV (5). C'est le même code que la doc d'OpenCV mais avec de la gestion d'erreur en plus. La fonction match_features se découpe en plusieurs parties :

- la recherche des descripteurs des points d'intérêt dans les 2 images.
- la comparaison des descripteurs pour retrouver les points d'intérêts d'une image dans l'autre.
- filtrage des points communs en fonction de la distance entre ces points dans les 2 images grâce à un coefficient.

```python
def match_features(img1_path, img2_path):
    # 1. Chargement des images
    img1 = cv.imread(img1_path)
    img2 = cv.imread(img2_path)

    # Vérification du chargement correct des images
    if img1 is None or img2 is None:
        print("Erreur: Impossible de charger une ou les deux images")
        return

    # 2. Conversion en niveaux de gris
    gray1 = cv.cvtColor(img1, cv.COLOR_BGR2GRAY)
    gray2 = cv.cvtColor(img2, cv.COLOR_BGR2GRAY)

    # 3. Détection des points d'intérêt et calcul des descripteurs
    sift = cv.SIFT_create()
    keypoints1, descriptors1 = sift.detectAndCompute(gray1, None)
    keypoints2, descriptors2 = sift.detectAndCompute(gray2, None)

    # Vérification si les descripteurs sont vides
    if descriptors1 is None or descriptors2 is None or len(descriptors1) == 0 or len(descriptors2) == 0:
        print("Erreur: Impossible de trouver des descripteurs dans une ou les deux images")
        return

    # Vérification supplémentaire: les descripteurs doivent être du même type
    if descriptors1.dtype != np.float32:
        descriptors1 = np.float32(descriptors1)
    if descriptors2.dtype != np.float32:
        descriptors2 = np.float32(descriptors2)

    # 4. Matching des descripteurs
    # Utilisation de BFMatcher au lieu de FLANN pour plus de robustesse
    bf = cv.BFMatcher()

    # Utilisez le matcher approprié et avec gestion d'erreurs
    try:
        # Vérifiez si nous avons assez de descripteurs pour kNN avec k=2
        k = min(2, len(descriptors2))
        matches = bf.knnMatch(descriptors1, descriptors2, k=k)

        # 5. Filtrage des bons matchs avec le test de ratio de Lowe (uniquement si k=2)
        good_matches = []
        if k == 2:
            for pair in matches:
                if len(pair) == 2:  # S'assurer que nous avons bien deux matches
                    m, n = pair
                    if m.distance < 1* n.distance:
                        good_matches.append(m)
        else:
            # Si k=1, prenez simplement tous les matches
            good_matches = [m[0] for m in matches if len(m) > 0]

    except cv.error as e:
        print(f"Erreur lors du matching: {e}")
        # Solution alternative: utilisation de matcher.match() au lieu de knnMatch
        matches = bf.match(descriptors1, descriptors2)
        good_matches = sorted(matches, key=lambda x: x.distance)[:30]  # Prendre les 30 meilleurs

    # 6. Affichage des résultats
    img_matches = cv.drawMatches(img1, keypoints1, img2, keypoints2, good_matches, None,
                                 flags=cv.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)

    # Conversion BGR->RGB pour matplotlib
    img_matches = cv.cvtColor(img_matches, cv.COLOR_BGR2RGB)

    cv.imshow("t",img_matches)
    cv.waitKey(0)
    cv.destroyAllWindows()

    return keypoints1, keypoints2, good_matches
```

Code 5 : 1ère fonction utillisé pour detecter les matchs de point d'interet

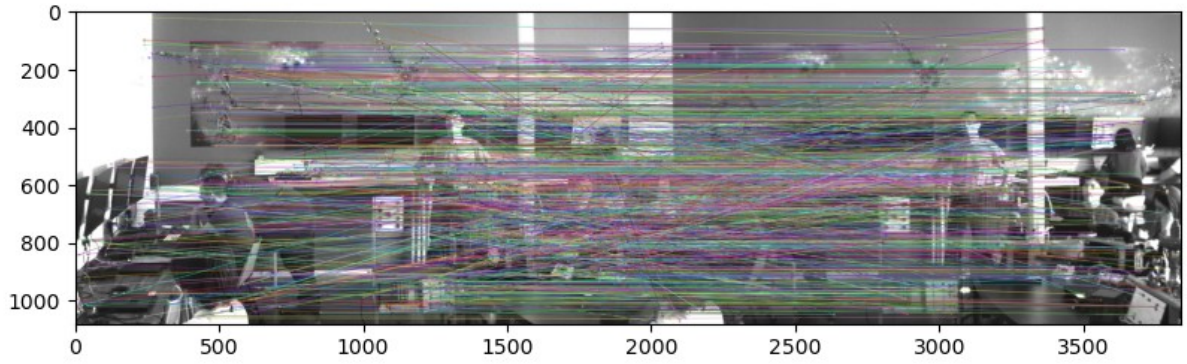

Fig 5 : screen du programme de Brute-Force Matching with SIFT Descriptors and Ratio Test

Dans le screen ci-dessus on observe qu'il y a 1156 points d'intérêt qui on été retenus par le programme mais que certains de ces points ne sont pas tous au même endroit dans les 2 images. Je pense que c'est dû au knn qui matche les features, certains doivent être proches dans leurs descriptions malgré qu'elles soient loin l'une de l'autre.

Ensuite, je me suis dit qu'il fallait undistort les images afin d'enlever la déformation causée par la caméra avant. Dans le screen ci-dessous (Fig 6), on voit que undistort les images ne change pas grand-chose.

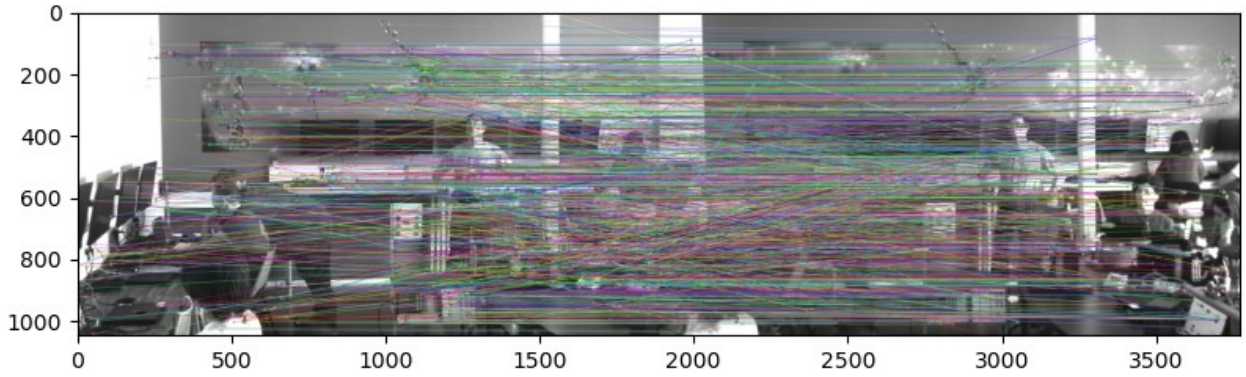

Fig 6 : screen du programme de Brute-Force Matching with SIFT Descriptors and Ratio Test avec les images undistort

V/ triangulation des points d'interets

Dans les parties suivantes j'ai gardé les matrices et vecteurs issus de la calibration en pixel afin de ne pas avoir de problème d'unité.

Pour la triangulation des points, j'utilise la fonction DLT (Direct Linear Transform) présente dans le repo (4), De ce que j'ai compris, on se sert des coordonnées d'un point d'intérêt présent dans les deux images (point1, point2), et des matrices de transformations (P1, P2) afin d'écrire une équation linéaire $Ax=0$ où x est le vecteur $[x,y,z,w]^T$ que l'on cherche.

On a besoin de deux caméras car avec une seule on n'a pas assez d'équations pour trouver l'inconnue x. Puis le résultat faut le diviser par w pour qu'il soit à l'échelle 1.

```
def DLT(P1, P2, point1, point2):

    A = [point1[1]*P1[2,:] - P1[1,:],
         P1[0,:] - point1[0]*P1[2,:],
         point2[1]*P2[2,:] - P2[1,:],
         P2[0,:] - point2[0]*P2[2,:]
        ]
    A = np.array(A).reshape((4,4))
    #print('A: ')
    #print(A)

    B = A.transpose() @ A
    from scipy import linalg
    U, s, Vh = linalg.svd(B, full_matrices = False)

    print('Triangulated point: ')
    print(Vh[3,0:3]/Vh[3,3])
    return Vh[3,0:3]/Vh[3,3]
```

Code 6 : triangulation des points d'interets

VI/ affichage de la réprésentation 3D de l'objet

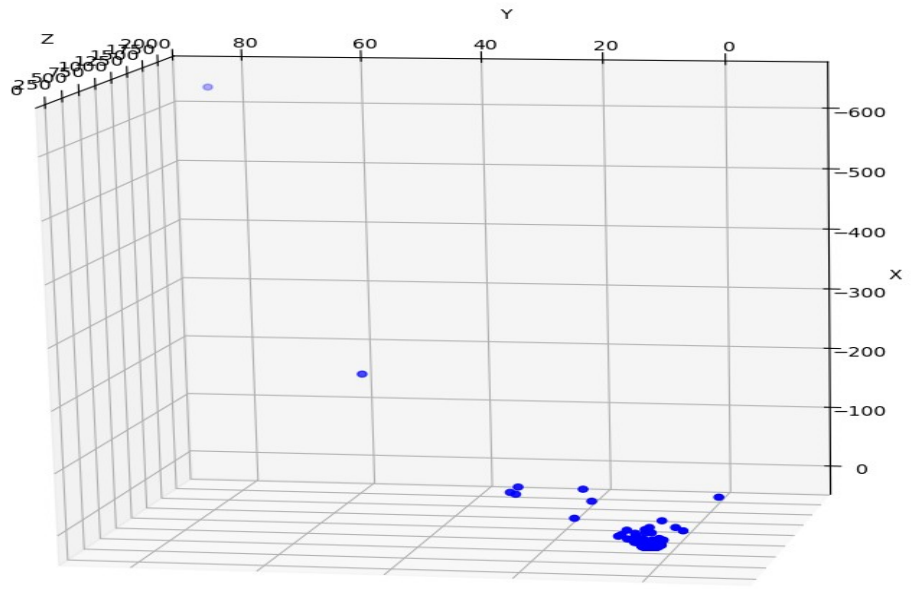

Fig 7 : 2ème tentative de representation 3D de la salle

Dans l'image ci-dessus (Fig 7), on observe que les données ressemblent à un plan en 2D qui a été incliné, et aussi ça ressemble à un des coins de la salle D217 qui est présent sur les photos de test. Je pense qu'il doit y avoir un problème dans mon code, normalement les matrices sont toutes en pixel donc je ne pense pas que ce soit un problème d'unité.

On observe dans les images ci-dessous que la fonction DLT à changé les positions x, y des points (Fig 8, Fig 9) et que la fonction match_features (Code 5) n'a pas bien matché certains points ensemble, donc ça peut être une des causes de la triangulation qui donne des valeurs qui sont soit éloignées soit condensées.

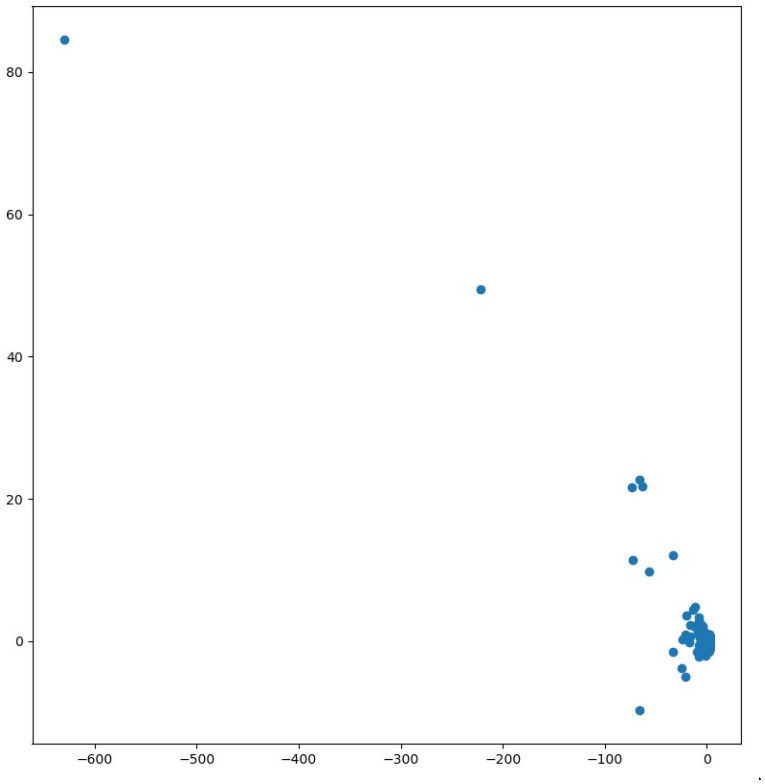

Fig 8 : screen du graphique montrant les coordonnées x,y en sortie du DLT

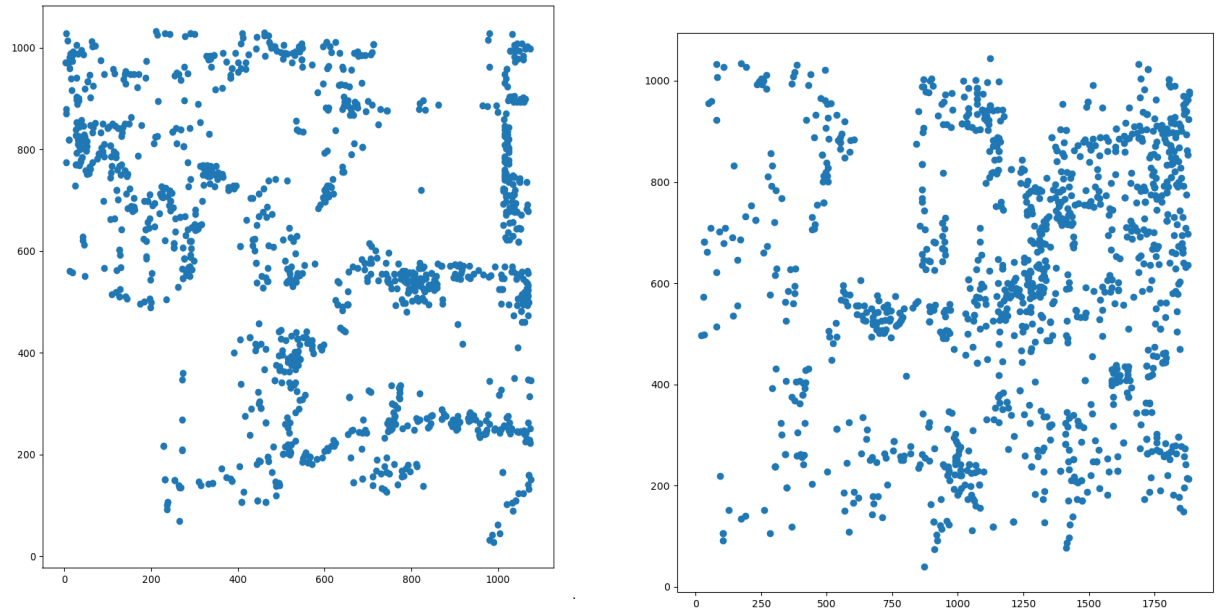

Fig 9 : screen montrant les point d'interet x,y dans les 2 images (caméra 1 à gauche, caméra 2 à droite)

Conclusion :

En effectuant ce projet j'ai compris la démarche à suivre pour faire une reconstruction 3D d'un objet dans une image en utilisant 2 images.

Ce que j'ai le mieux compris c'est la calibration mono et stéréo des caméras avec OpenCV en Python.

Pour moi le plus compliqué c'est la triangulation des points, il me manque encore quelques infos pour bien comprendre comment faire et arriver à afficher la représentation 3D d'un objet. J'ai tenté d'utiliser le moins possible ChatGPT/Claude mais ils m'ont bien aidé pour comprendre comment certaines documentations et morceaux de code fonctionnent.

Sources utilisé durant le projet :

(1) https://s1-dl.theimagingsource.com/api/2.5/packages/documentation/manual-trm/trmdmk33gx174/4bf7d60f-47bc-5495-81b4-85d8fc99d1f1/trmdmk33gx174.en_US.pdf

(2) https://lazka.github.io/pgi-docs/Aravis-0.8/index.html

(3) https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html

(4) https://temugeb.github.io/opencv/python/2021/02/02/stereo-camera-calibration-and-triangulation.html

(5) https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html