HW2

姓名:簡晟棋
學號:r14922022

Environment

Python == 3.11.0

opencv-python == 4.12.0.88

numpy == 2.2.6

open3d == 0.19.0

pandas == 2.3.2

scipy == 1.16.2

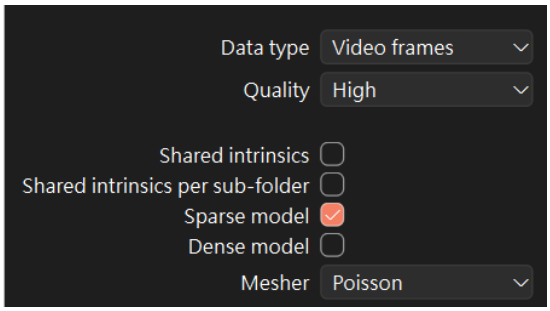
tqdm == 4.67.1

Problem1

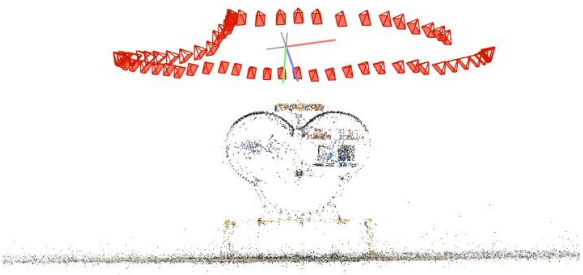
1-1

使用 splitvideo.py 將影片每 1 秒擷取 5 frame，輸入到 colmap 用 automatic reconstruction

設定:



result:

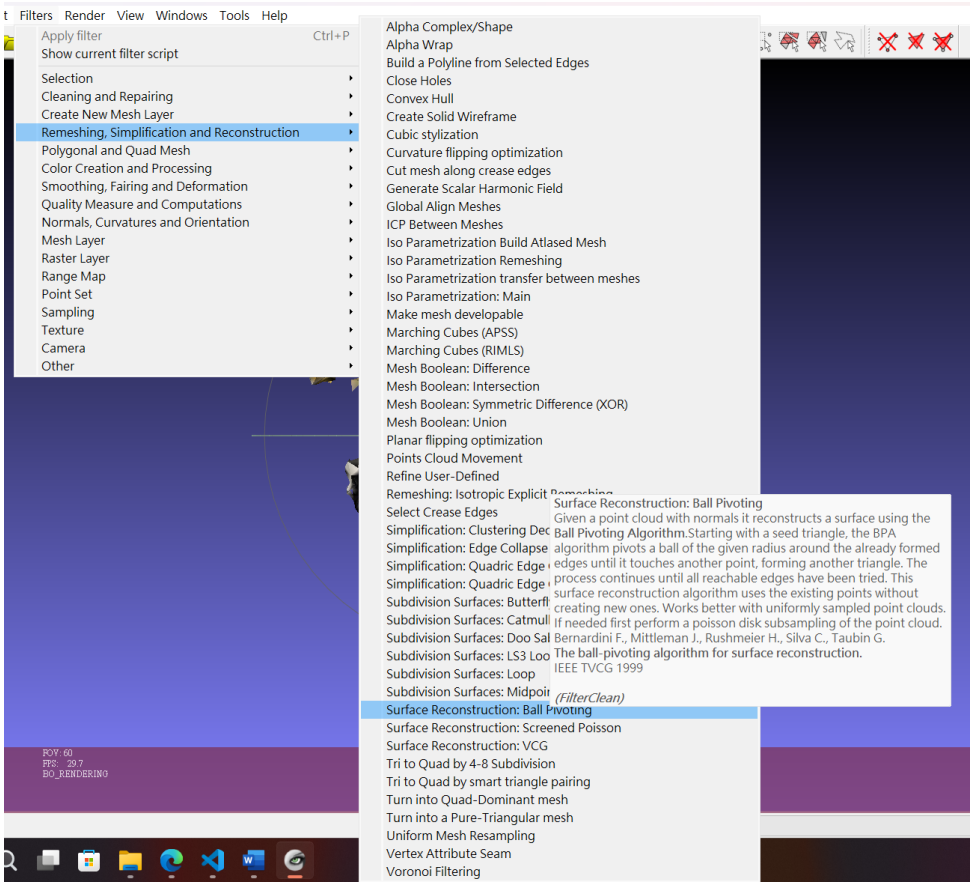


1-2

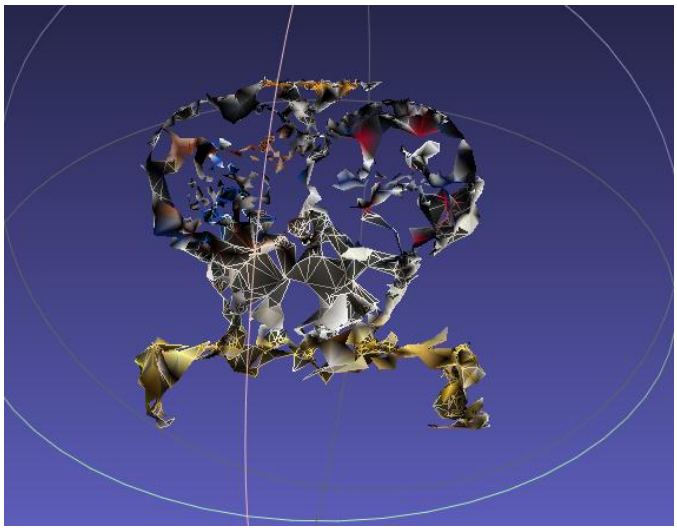
用 meshlab 重建

1.把多餘的點手動清除

2.使用 Ball Pivoting 重建 surface



result:



觀察:sparse model 的點較為稀疏，重建有一定難度

影片連結:https://www.youtube.com/watch?v=kS03sSNckk0

Problem2

使用 2d3dmathcing.py 完成 2-1 與 2-2

2-1

Step1:算 world to camera [R|t]

```
def pnpsolver(query,model,cameraMatrix=0,distortion=0,self_flag = False):
    kp_query, desc_query = query
    kp_model, desc_model = model
    cameraMatrix = np.array([[1868.27,0,540],[0,1869.18,960],[0,0,1]])
    distCoeffs = np.array([0.0847023,-0.192929,-0.000201144,-0.000725352])

    # TODO: solve PnP problem using OpenCV
    # Hint: you may use "Descriptors Matching and ratio test" first
    matcher = cv2.BFMatcher()
    matches = matcher.knnMatch(desc_query, desc_model, k=2)
    good_matches = []
    for m, n in matches:
        if m.distance < 0.75 * n.distance:
            good_matches.append(m)

    good_matches = sorted(good_matches, key=lambda x: x.distance)
    pointsquery = np.array([kp_query[m.queryIdx] for m in good_matches])
    pointsmodel = np.array([kp_model[m.trainIdx] for m in good_matches])
```

用 hw1 的 match 方法選出 match 的 2D 點與 3D 點

```
self_flag = True
for i in tqdm(IMAGE_ID_LIST):
    # Load quaery image
    fname = f"valid_img{i*5}.jpg"
    idx = images_df.loc[images_df["NAME"] == fname]["IMAGE_ID"].values[0]
    rimg = cv2.imread("data/frames/" + fname)

    # Load query keypoints and descriptors
    points = point_desc_df.loc[point_desc_df["IMAGE_ID"] == idx]
    kp_query = np.array(points["XY"].to_list())
    desc_query = np.array(points["DESCRIPTORS"].to_list()).astype(np.float32)

    # Find correspondance and solve pnp
    retval, rvec, tvec, inliers = pnpsolver((kp_query, desc_query), (kp_model, desc_model),self_flag=self_flag)
    if not retval:
        continue
```

```
    if not self_flag:
        return cv2.solvePnPRansac(
            pointsmodel, pointsquery, cameraMatrix, distCoeffs,
            iterationsCount=100, reprojectionError=8.0, confidence=0.99
        )
    return ransac_p3p(pointsmodel, pointsquery, cameraMatrix, distCoeffs)
```

用 self_flag 決定用自己的 p3p_ransac 還是 opencv 的 solvePnPRansac function 解出[R|t]

自己實作 p3p(distortion 部分用 chatGPT 得出):

```python
def World2Plane(world_pts, r, t, K, distCoeffs):

    #world to camera
    pts_cam = (r @ world_pts.T + t.reshape(3,1)).T

    #camera normalize
    x = pts_cam[:,0] / pts_cam[:,2]
    y = pts_cam[:,1] / pts_cam[:,2]

    #distortion
    k1, k2, p1, p2 = distCoeffs
    r2 = x**2 + y**2
    radial = 1 + k1*r2 + k2*r2**2
    x_dist = x * radial + 2*p1*x*y + p2*(r2 + 2*x**2)
    y_dist = y * radial + p1*(r2 + 2*y**2) + 2*p2*x*y

    u = K[0,0]*x_dist + K[0,2]
    v = K[1,1]*y_dist + K[1,2]
    return np.vstack([u, v]).T #(points number, 2)
```

World2Plane:

1.把 world 座標轉乘 camera 座標

2.camera 座標標準化

3.用 k1,k2,p1,p2 與 instinct matrix 得到 distortion 後的 plane 座標

```python
def undistort_points(pts, K, distCoeffs, max_iter=5):
    fx, fy = K[0,0], K[1,1]
    cx, cy = K[0,2], K[1,2]
    k1, k2, p1, p2 = distCoeffs

    undistorted = []
    for u, v in pts:#iteration for points
        #normalize
        x = (u - cx) / fx
        y = (v - cy) / fy
        x_u, y_u = x, y
        for _ in range(max_iter):#iteration solve undistort
            r2 = x_u**2 + y_u**2
            radial = 1 + k1*r2 + k2*r2**2
            x_u = (x - 2*p1*x_u*y_u - p2*(r2 + 2*x_u**2)) / radial
            y_u = (y - p1*(r2 + 2*y_u**2) - 2*p2*x_u*y_u) / radial
        undistorted.append([x_u, y_u, 1.0])
    return np.array(undistorted) / np.linalg.norm(undistorted, axis=1).reshape(-1,1)
    #(points number, 3)
    #normalize for p3p direction
```

undistort_points:

1.用 iteration 的方式估計 undistorted 的 camera 座標

2.將 camera 座標 normalize 成單位方向，p3p 要用

```python
def p3p_solver(world_pts, cam_dirs):

    poses = []

    reindex = None
    perms = list(itertools.permutations([0,1,2]))

    for perm in perms:
        i1,i2,i3 = perm
        if np.dot(cam_dirs[i1],cam_dirs[i3]) <= np.dot(cam_dirs[i1],cam_dirs[i2]) \
            and np.dot(cam_dirs[i1],cam_dirs[i2]) <= np.dot(cam_dirs[i2],cam_dirs[i3]):
            reindex = list(perm)
            break

    m1, m2, m3 = cam_dirs[reindex]

    m12 = np.dot(m1,m2)
    m13 = np.dot(m1,m3)
    m23 = np.dot(m2,m3)

    X1, X2, X3 = world_pts[reindex]
    X1 = X1.reshape(1,3)
    X2 = X2.reshape(1,3)
    X3 = X3.reshape(1,3)

    m1 = m1.reshape(1,3)
    m2 = m2.reshape(1,3)
    m3 = m3.reshape(1,3)

    s12 = np.sum((X1-X2)**2)
    s23 = np.sum((X2-X3)**2)
    s13 = np.sum((X1-X3)**2)
```

```python
    c4 = -s12**2 + 2*s12*s13 + 2*s12*s23 - s13**2 \
        + 4*s13*s23*m12**2 - 2*s13*s23 - s23**2
    c3 = 4*s12**2*m13 - 4*s12*s13*m12*m23 - 4*s12*s13*m13\
        - 8*s12*s23*m13 + 4*s13**2*m12*m23\
        - 8*s13*s23*m12**2*m13 - 4*s13*s23*m12*m23\
        + 4*s13*s23*m13 + 4*s23**2*m13
    c2 = -4*s12**2*m13**2 - 2*s12**2 + 8*s12*s13*m12*m13*m23\
        + 4*s12*s13*m23**2 + 8*s12*s23*m13**2 + 4*s12*s23\
        - 4*s13**2*m12**2 - 4*s13**2*m23**2 + 2*s13**2\
        + 4*s13*s23*m12**2 + 8*s13*s23*m12*m13*m23\
        - 4*s23**2*m13**2 - 2*s23**2
    c1 = 4*s12**2*m13 - 4*s12*s13*m12*m23\
        - 8*s12*s13*m13*m23**2 + 4*s12*s13*m13\
        - 8*s12*s23*m13 + 4*s13**2*m12*m23\
        - 4*s13*s23*m12*m23 - 4*s13*s23*m13\
        + 4*s23**2*m13
    c0 = -s12**2 + 4*s12*s13*m23**2 - 2*s12*s13\
        + 2*s12*s23 - s13**2 + 2*s13*s23 - s23**2
    coeffs = [c4, c3, c2, c1, c0]
    roots = np.roots(coeffs)
    roots = [r.real for r in roots if np.isreal(r) and r.real > 0]
    A = -s12+s23+s13
    B = 2*(s12-s23)*m13
    C = -s12+s23-s13

    for x in roots:
        y = (A*x**2 + B*x + C) / (2*s13*(m12*x - m23))
        if y>0:
            d3 = np.sqrt(s12/(x**2-2*x*y*m12+y**2))
            d1 = x*d3
            d2 = y*d3
            X = np.hstack([X1.T - X2.T , X1.T - X3.T , np.cross(X1-X2,X1-X3).T])
            Y1 = d1*m1 - d2*m2
            Y2 = d1*m1 - d3*m3
            Y = np.hstack([Y1.T,Y2.T,np.cross(Y1,Y2).T])
            r = Y @ np.linalg.inv(X)
            t = d1*m1.T - r@X1.T
            poses.append((r,t))
    return poses
```

p3p_solver:

根據 P3P Made Easy，算出 world to camera 的 R,t

為了簡化運算，paper 中 refine 深度的部分用 ransac 取代

```python
def ransac_p3p(world_pts, image_pts, K, distCoeffs,
               iterations=100, threshold=8, confidence=0.99):

    best_inliers = []
    best_pose = None
    dirs = undistort_points(image_pts, K, distCoeffs)
    N = world_pts.shape[0]
    best_error = np.inf

    for _ in range(iterations):#N iterations
        idx = np.random.choice(N, 3, replace=False)#sample S point
        sols = p3p_solver(world_pts[idx], dirs[idx])#fit
        if len(sols) == 0:
            continue

        for r, t in sols:
            proj = World2Plane(world_pts, r, t, K, distCoeffs)
            err = np.linalg.norm(proj - image_pts, axis=1)
            inliers = np.where(err < threshold)[0]#threshold d
            if len(inliers) >= N*confidence:#good fit
                err = np.mean(np.linalg.norm(proj[inliers] - image_pts[inliers], axis=1))
                if err<best_error:
                    best_inliers = inliers
                    best_pose = (r, t)
                    best_error = err

    if best_pose is None:
        return False, None, None, None

    r, t = best_pose
    rvec = R.from_matrix(r).as_rotvec()
    tvec = t.reshape(3,1)
    inliers = np.array(best_inliers).reshape(-1,1)

    return True, rvec, tvec, inliers
```

ransac_p3p:

實作 ransac

在多次 iteraion 中

1.每次隨機選 3 組點用 p3p_solver 去 fit

2.fit 結果用來把 world 座標用 World2Plane 得到 plane 座標

3.算有多少點誤差在 threshold 內

4.如果在 threshold 內的 inliner 數量>總數*confidence，則是 good fit

回傳最好的 fit 結果

Step2:error 計算

```python
def rotation_error(R1, R2):
    #TODO: calculate rotation error
    R1 = R.from_quat(R1)
    R2 = R.from_quat(R2)
    R_rel = R1 * R2.inv()
    rotvecs = R_rel.as_rotvec()
    return np.median(rotvecs)

def translation_error(t1, t2):
    #TODO: calculate translation error
    return np.sqrt(np.sum((t1 - t2) ** 2))
```

rotation error:

把 R1 與 R2 的 inverse 相乘後轉乘 rvec 形式，取中位數

translation error:

算 t1,t2 的 2-norm distance

Step3:畫相機位置與軌跡

```python
Camera2World_Transform_Matrixs = []
for r, t in zip(r_list, t_list):
    # TODO: calculate camera pose in world coordinate system
    c2w = np.zeros((3,4))
    c2w[:3,:3] = R.from_rotvec(r.reshape(1,3)).as_matrix().T.reshape(3,3)
    c2w[:3,3] = (-c2w[:3,:3] @ t.reshape(3,1)).flatten()
    Camera2World_Transform_Matrixs.append(c2w)

visualization(Camera2World_Transform_Matrixs, points3D_df)
```

先把 world_to_camera matrix 轉成 camera_to_world matrix

$Xw = R^{-1} * Xc - R^{-1} * t$

```python
s = 0.05
f = 0.1
track = []
for c2w in Camera2World_Transform_Matrixs:
    points = np.array([
        [0, 0, 0,1],
        [-s, -s, f,1],
        [ s, -s, f,1],
        [ s,  s, f,1],
        [-s,  s, f,1]
    ])

    points = (c2w @ points.T).T
    track.append(points[0])
    lines = [
        [0, 1], [0, 2], [0, 3], [0, 4],
        [1, 2], [2, 3], [3, 4], [4, 1]
    ]
    camera_pyramid = o3d.geometry.LineSet(
        points=o3d.utility.Vector3dVector(points),
        lines=o3d.utility.Vector2iVector(lines),
    )
    camera_pyramid.paint_uniform_color([1, 0, 0])


    vis.add_geometry(camera_pyramid)
```

```python
track_lines = [[i,i+1] for i in range(len(track)-1)]

trackline = o3d.geometry.LineSet(
    points=o3d.utility.Vector3dVector(track),
    lines=o3d.utility.Vector2iVector(track_lines),
)
trackline.paint_uniform_color([0, 0, 0])
vis.add_geometry(trackline)
```

visualization:

用 camera_to_world matrix 轉換角錐點(points)位置後，用 points 與 lines 畫一個四角椎 camera_pyramid，角錐頂點則一個一個連成線(trackline)

result:

pnp+ransac with opencv:

rotation error: 8.942008584830542e-07

translation error: 0.00016467364717986384

手刻 p3p+ransac:

rotation error: 7.630770265330595e-06

translation error: 0.0019594450789294283

2-2

```python
box_maxtrix = np.load("cube_transform_mat.npy")
box_point = np.array([[0,0,0,1],
                      [1,0,0,1],
                      [0,1,0,1],
                      [1,1,0,1],
                      [0,0,1,1],
                      [1,0,1,1],
                      [0,1,1,1],
                      [1,1,1,1]])
box_point = (box_maxtrix @ box_point.T).T
box_point = np.hstack([box_point,np.ones((box_point.shape[0],1))])
```

```python
faces = [
    [0,1,3,2],
    [4,5,7,6],
    [0,1,5,4],
    [2,3,7,6],
    [0,2,6,4],
    [1,3,7,5]
]


colors = [
    (255, 0, 0),    # Blue
    (0, 255, 0),    # Green
    (0, 0, 255),    # Red
    (255, 255, 0), # Cyan
    (255, 0, 255), # Magenta
    (0, 255, 255)  # Yellow
]
```

將 box point 的 8 個點，每面需要的點與顏色設定好

8 個點為 1*1*1 的立方體乘上 cube_transform_mat

```python
N = 30
x = np.arange(N)
y = np.arange(N)
X, Y = np.meshgrid(x, y, indexing='xy')
X, Y = X.reshape(-1,1)/N, Y.reshape(-1,1)/N
weights = np.hstack([1-X-Y,X,np.zeros(X.shape),Y])
```

```python
w2c = np.zeros((3,4))
w2c[:3,:3] = R.from_rotvec(rvec.reshape(1,3)).as_matrix()
w2c[:3,3] = tvec.flatten()
box_point_camera = (cameraMatrix @ w2c @ box_point.T).T
box_point_plane = (box_point_camera[:,:2] / box_point_camera[:,2].reshape(-1,1)).astype(np.int32)



point_depths = []


for j, face in enumerate(faces):
    face_points_camera = weights @ box_point_camera[face]
    face_points_plane = (face_points_camera[:,:2] / face_points_camera[:,2].reshape(-1,1)).astype(np.int32)
    for k in range(face_points_camera.shape[0]):
        if face_points_camera[k,2]>0:
            point_depths.append((face_points_camera[k,2],face_points_plane[k],j))

point_depths = sorted(point_depths,key=lambda x: x[0],reverse=True)


for _,point,j in point_depths:
    cv2.circle(rimg, point, 10, colors[j], -1)
```

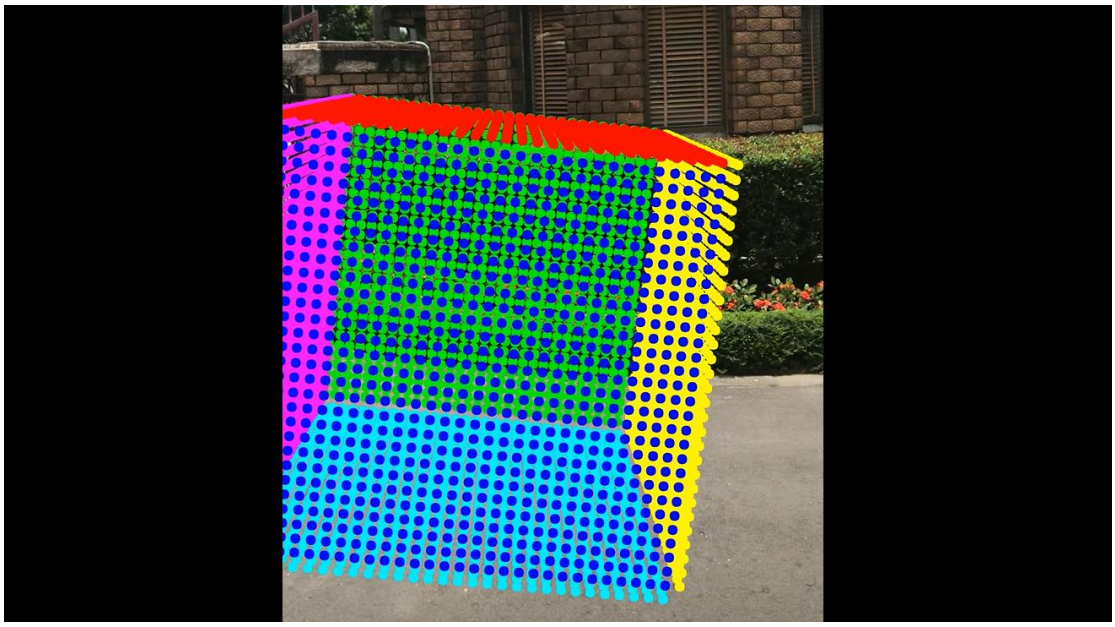
1.用 world_to_camera matrix 與 instinct matrix of camera 把 box point 的點投射到 plane 上

2.接著用 meshgrid 得出每個面的點所需要的四頂點 weights，將每一面的平行四邊形分成 30*30 個點

3.根據點的深度做排序，從深度高到低一點一點畫上去，並排除深度<0(在相機後面)的點

方塊位置:

result(影片截圖):



影片連結: https://youtu.be/XtT7NHUgbVY