Computer Vision Homework2

112062636 游竣量

Fundamental Matrix Estimation from Point Correspondences(Question 1)

Implementation:

(a.)

$$(uu', uv', u, vu', vv', v, u', v', 1)\begin{pmatrix} F_{11} \\ F_{12} \\ F_{13} \\ F_{21} \\ F_{22} \\ F_{23} \\ F_{31} \\ F_{32} \\ F_{33} \end{pmatrix} = 0$$

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import os

# 指定資料夾路徑
dir_name = "./assets/"
output_dir = "./output/"
# 建立圖片完整路徑
img1_path = os.path.join(dir_name, "image1.jpg")
img2_path = os.path.join(dir_name, "image2.jpg")
# 讀取圖片
img1 = cv2.imread(img1_path)
img2 = cv2.imread(img2_path)
# 建立座標檔案完整路徑
pt1_path = os.path.join(dir_name, "pt_2D_1.txt")
pt2_path = os.path.join(dir_name, "pt_2D_2.txt")
```

```python
# 初始化座標列表
coordinates = []
# 打開第一個點座標檔案
with open(pt1_path, "r") as file1:
    # 打開第二個點座標檔案
    with open(pt2_path, "r") as file2:
        # 讀取點的數量
        num_lines = int(file1.readline().strip())
        # 跳過第二個檔案的點數量行
        file2.readline()
        # 對每一對座標進行迴圈
        for i in range(num_lines):
            # 讀取對應的座標點，將其從字符串轉換為浮點數列表
            coordinate = list(
                map(
                    float,
                    file1.readline().strip().split() +
file2.readline().strip().split(),
                )
            )
            # 將座標點添加到列表中
            coordinates.append(coordinate)
# 將座標列表轉換為 numpy 數組
coordinates = np.array(coordinates)


def generate_matrix_A(coordinates):
    # 從座標數組中分別提取出對應點的 x1, y1, x2, y2 座標
    x1, y1, x2, y2 = (
        coordinates[:, 0],  # 第一張圖片中點的 x 座標
        coordinates[:, 1],  # 第一張圖片中點的 y 座標
        coordinates[:, 2],  # 第二張圖片中點的 x 座標
        coordinates[:, 3],  # 第二張圖片中點的 y 座標
    )

    # 根據八點算法構建矩陣 A 的每一列，這裡使用的是展開後的對應點座標乘積
    # 每個乘積形成 A 矩陣的一行
    return np.column_stack(
        [x1 * x2, x1 * y2, x1, y1 * x2, y1 * y2, y1, x2, y2, np.ones_like(x1)]
```

```python
    )


def SVD_4_F_rank2(A):
    # 使用奇異值分解（SVD）來解決方程式 AF = 0
    # 對 A 矩陣進行 SVD 分解，得到 U, D, VT，其中 A = UDV^T
    # 這裡的 U 是左奇異向量，D 是奇異值對角矩陣，VT（V 的轉置）是右奇異向量
    U, D, VT = np.linalg.svd(A)
    # 從 VT（V 的轉置）中提取最後一行，這是與最小奇異值對應的向量
    # 這個向量（F 的元素）被重新排列成 3x3 的矩陣形式，得到初始的基本矩陣 F
    F = VT[-1].reshape(3, 3)

    # 將 F 矩陣做 SVD 分解
    U, D, VT = np.linalg.svd(F)
    # 在 F 上強制執行 rank2 約束
    # 設置 D 中的最小奇異值為 0，這是為了保證 F 為 rank2
    D[2] = 0
    # 使用更新後的奇異值和原始的 U 和 VT 重新構造基本矩陣 F
    # 得到滿足 rank2 約束的基本矩陣 rank2_F
    rank2_F = np.dot(U, np.dot(np.diag(D), VT))

    return rank2_F


matrix_A = generate_matrix_A(coordinates)
rank2_F = SVD_4_F_rank2(matrix_A)
print(rank2_F)
```

```
[[ 5.73019754e-07  1.84317952e-06 -3.95680928e-04]
 [ 3.49579399e-06  1.02102047e-06  5.43607666e-03]
 [-2.91687458e-03 -8.06156716e-03  9.99948396e-01]]
```

(b.)

```
def normalize(coordinates):
    # 計算第一組座標（圖像 1 中的點）的平均值
    mean1 = np.mean(coordinates[:, :2], axis=0)
    # 計算第二組座標（圖像 2 中的點）的平均值
    mean2 = np.mean(coordinates[:, 2:], axis=0)
    # 計算第一組座標點到其平均值的平均歐氏距離
    dist1 = np.sqrt(np.sum((coordinates[:, :2] - mean1) ** 2, axis=1)).mean()
    # 計算第二組座標點到其平均值的平均歐氏距離
    dist2 = np.sqrt(np.sum((coordinates[:, 2:] - mean2) ** 2, axis=1)).mean()
    # 計算第一組座標點的縮放因子，以使平均距離成為 sqrt(2)
    scale1 = np.sqrt(2) / dist1
    # 計算第二組座標點的縮放因子，以使平均距離成為 sqrt(2)
    scale2 = np.sqrt(2) / dist2
    # 構造第一組座標點的正規化轉移矩陣
    T1 = np.array(
        [
            [scale1, 0, -scale1 * mean1[0]],
            [0, scale1, -scale1 * mean1[1]],
            [0, 0, 1],
        ]
    )
    # 構造第二組座標點的正規化轉移矩陣
    T2 = np.array(
        [
            [scale2, 0, -scale2 * mean2[0]],
            [0, scale2, -scale2 * mean2[1]],
            [0, 0, 1],
        ]
    )
    # 對第一組座標點應用轉移矩陣，進行正規化處理
    normalized_coords1 = np.dot(
        T1, np.column_stack((coordinates[:, :2],
np.ones(coordinates.shape[0])))).T
    ).T
    # 對第二組座標點應用轉移矩陣，進行正規化處理
    normalized_coords2 = np.dot(
```

```python
        T2, np.column_stack((coordinates[:, 2:],
np.ones(coordinates.shape[0]))).T
    ).T
    # 組合兩組正規化後的座標點
    normalized_coordinates = np.column_stack(
        (normalized_coords1[:, :2], normalized_coords2[:, :2])
    )

    # 返回正規化後的座標點和兩個轉移矩陣
    return normalized_coordinates, T1, T2


# 將座標點正規化，以提高數值穩定性和計算精度
normalized_coordinates, transform_matrix1, transform_matrix2 =
normalize(coordinates)
# 使用正規化後的座標點生成對應的矩陣 A
normalized_matrix_A = generate_matrix_A(normalized_coordinates)
# 使用 SVD 方法從正規化後的矩陣 A 中提取出 rank2 的基本矩陣 F
normalized_rank2_F = SVD_4_F_rank2(normalized_matrix_A)
# 將正規化基本矩陣 F 反正規化，將其轉換回原始座標系的尺度
denormalized_rank2_F = np.dot(
    transform_matrix1.T, np.dot(normalized_rank2_F, transform_matrix2)
)
print(denormalized_rank2_F)
```

```
[[ 1.21520423e-07  6.47004074e-07  2.92789910e-04]
 [ 1.09357327e-06 -4.35355028e-08  5.01038256e-03]
 [-4.91106886e-04 -5.70144542e-03  6.67389392e-02]]
```

(c-1.)

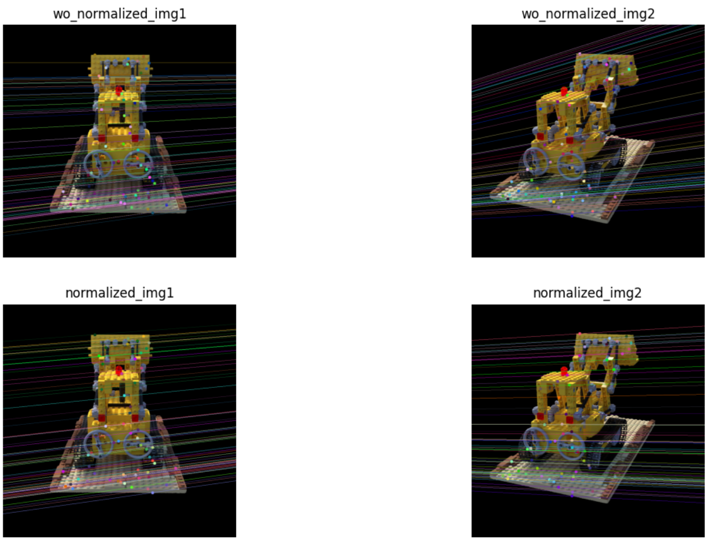
$\mathcal{F}$ p′ is the epipolar line associated with p′.

$\mathcal{F}^T$ p is the epipolar line associated with p.

```
# 分割座標對、並加入全為 1 的行，形成齊次座標
coords1 = np.column_stack((coordinates[:, :2],
np.ones(coordinates.shape[0])))
coords2 = np.column_stack((coordinates[:, 2:],
np.ones(coordinates.shape[0])))

# 利用基本矩陣 'rank2_F' 計算第二幅影像中的點在第一幅影像上對應的外極線
lines1 = np.dot(rank2_F, coords2.T).T
# 利用基本矩陣 'rank2_F' 的轉置，計算第一幅影像中的點在第二幅影像上對應的外極線
lines2 = np.dot(rank2_F.T, coords1.T).T
# 同上，改成正規化版
normalized_lines1 = np.dot(denormalized_rank2_F, coords2.T).T
normalized_lines2 = np.dot(denormalized_rank2_F.T, coords1.T).T


def draw_lines(img, lines, coords):
    # 對於傳入的每條線和對應的點座標
    for l, p in zip(lines, coords):
        # 隨機選擇一種顏色來繪製線和點
        color = tuple(np.random.randint(0, 255, 3).tolist())
        # 將點座標轉換為整數
        x, y, _ = map(int, p)
        # line ax + by + c = 0
        # 計算線在圖像邊界左邊的交點
        # x = 0，則 y = -c / b
        lx1, ly1 = map(int, [0, -l[2] / l[1]])
        # 計算線在圖像邊界右邊的交點
        # x = width，則 y = -(c + a * width) / b
        lx2, ly2 = map(int, [img.shape[1], -(l[2] + l[0] * img.shape[1]) /
l[1]])
        # 在圖像上繪製一條從左到右的線
        img = cv2.line(img, (lx1, ly1), (lx2, ly2), color, 1)
```

```python
        # 在對應點的位置繪製一個圓形標記
        img = cv2.circle(img, (x, y), 5, color, -1)

    # 將圖像從 BGR 色彩空間轉換到 RGB 色彩空間後返回
    return cv2.cvtColor(img, cv2.COLOR_BGR2RGB)


# 畫點跟線
wo_normalized_img1 = draw_lines(img1.copy(), lines1, coords1)
wo_normalized_img2 = draw_lines(img2.copy(), lines2, coords2)
normalized_img1 = draw_lines(img1.copy(), normalized_lines1, coords1)
normalized_img2 = draw_lines(img2.copy(), normalized_lines2, coords2)

# 如果 output_dir 不存在則新增一個
if not os.path.exists(output_dir):
    os.makedirs(output_dir)
# 存檔
cv2.imwrite(
    output_dir + "wo_normalized_img1.jpg",
    cv2.cvtColor(wo_normalized_img1, cv2.COLOR_RGB2BGR),
)
cv2.imwrite(
    output_dir + "wo_normalized_img2.jpg",
    cv2.cvtColor(wo_normalized_img2, cv2.COLOR_RGB2BGR),
)
cv2.imwrite(
    output_dir + "normalized_img1.jpg", cv2.cvtColor(normalized_img1,
cv2.COLOR_RGB2BGR)
)
cv2.imwrite(
    output_dir + "normalized_img2.jpg", cv2.cvtColor(normalized_img2,
cv2.COLOR_RGB2BGR)
)

# 將最後的 outputs 合併顯示
plt.figure(figsize=(12, 8))

plt.subplot(2, 2, 1)
plt.imshow(wo_normalized_img1)
```

```
plt.title("wo_normalized_img1")
plt.axis("off")

plt.subplot(2, 2, 2)
plt.imshow(wo_normalized_img2)
plt.title("wo_normalized_img2")
plt.axis("off")

plt.subplot(2, 2, 3)
plt.imshow(normalized_img1)
plt.title("normalized_img1")
plt.axis("off")


plt.subplot(2, 2, 4)
plt.imshow(normalized_img2)
plt.title("normalized_img2")
plt.axis("off")

plt.show()
```



(c-2.)

```python
def calculate_distance(points, lines):
    # 計算點到其對應 epipolar 線的距離，
    # 點到線的距離是通過公式 (ax + by + c) / sqrt(a^2 + b^2) 計算，其中[a, b, c]是線的參數，[x, y]是點的座標。
    distances = np.abs(lines[:, 0] * points[:, 0] + lines[:, 1] * points[:, 1] + lines[:, 2]) / np.sqrt(lines[:, 0]**2 + lines[:, 1]**2)
    # 計算距離的平均值。
    return np.mean(distances)

# 第一組座標（coords1）到其對應線（lines1）的平均距離。
distance1 = calculate_distance(coords1, lines1)
# 第二組座標（coords2）到其對應線（lines2）的平均距離。
distance2 = calculate_distance(coords2, lines2)

# 計算兩組平均距離的平均值。
average_distance = (distance1 + distance2) / 2
print(f"average_wo_normalized_distance {average_distance}")

# 同上，改成正規化版
normalized_distance1 = calculate_distance(coords1, normalized_lines1)
normalized_distance2 = calculate_distance(coords2, normalized_lines2)
average_normalized_distance = (normalized_distance1 + normalized_distance2) / 2
print(f"average_normalized_distance {average_normalized_distance}")
```

```
average_wo_normalized_distance 25.45418786366018
average_normalized_distance 0.9079239490639669
```

討論：

average_wo_normalized_distance 與 average_normalized_distance，分別對應到未正規化和正規化後的數據的基本矩陣算出之對應點與外極線平均誤差距離。

正規化後的數據的基本矩陣得出的平均距離更小，這表示正規化有助於提升基本矩陣的準確性。正規化過程通過降低計算誤差，使得特徵點與外極線的距離更為接近，從而提高了點的對應關係的精確度。，從結果圖也能明顯看出，數據經過正規化算出的外極線完美 match 對應點。

Homography transform: (Question 2)

Implementation:

(a.)

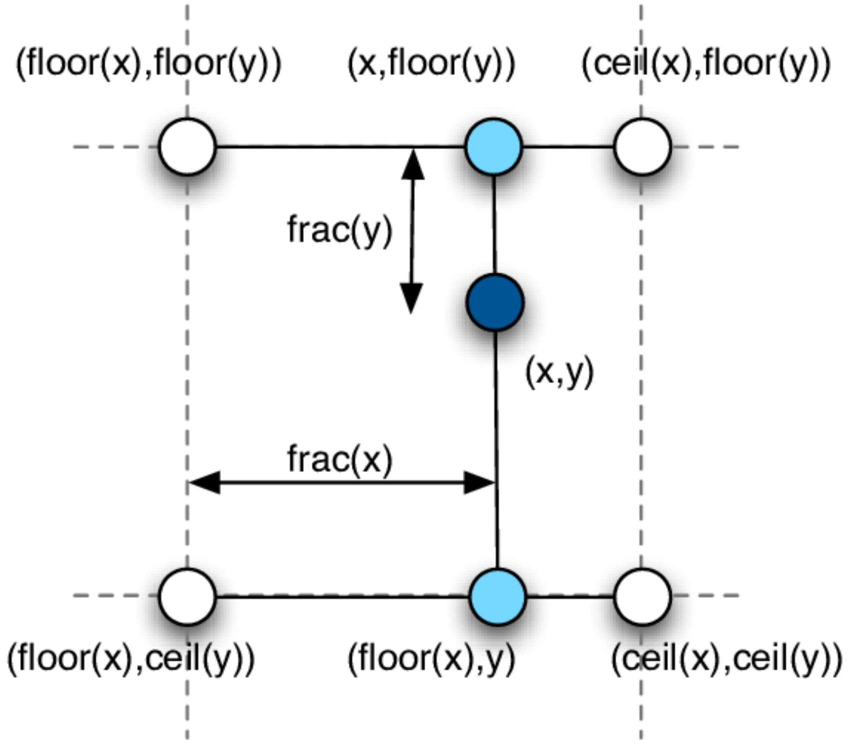
$$\begin{bmatrix} x_i & y_i & 1 & 0 & 0 & 0 & -x_i'x_i & -x_i'y_i & -x_i' \\ 0 & 0 & 0 & x_i & y_i & 1 & -y_i'x_i & -y_i'y_i & -y_i' \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

```
def Find_Homography(src, tar):
    A = []  # 初始化矩陣 A，用於構建線性方程組

    # 遍歷點對，構建方程組中的矩陣 A
    for p, p_ in zip(src, tar):
        x, y = p  # 源點座標
        x_, y_ = p_  # 目標點座標
        # 對於每個點對，根據單應性矩陣的定義構建兩行線性方程
        line1 = [x, y, 1, 0, 0, 0, -x_ * x, -x_ * y, -x_]
        A.append(line1)  # 添加到矩陣 A
        line2 = [0, 0, 0, x, y, 1, -y_ * x, -y_ * y, -y_]
        A.append(line2)  # 添加到矩陣 A

    A = np.array(A)  # 將 A 轉換為 numpy 陣列形式

    # 使用奇異值分解（SVD）解線性方程組
    U, D, VT = np.linalg.svd(A)
    H = VT[-1].reshape(3, 3)  # 解是 VT 的最後一行，並將其重新塑形為 3x3 矩陣

    return H  # 返回單應性矩陣 H
```

```
H = Find_Homography(corner_list_src, corner_list)
print(H)
```

```
[[3.84461016e-03 3.14250320e-05 9.76173483e-01]
 [1.19141107e-03 2.90750814e-03 2.16927441e-01]
 [2.29148885e-06 1.20195066e-07 1.77809378e-03]]
```

(b.)




```
def inverse_mapping(img_src, img_tar, H):
    # 計算變換矩陣 H 的逆矩陣 H_inv
    H_inv = np.linalg.inv(H)
    # 獲取目標影像 tar 的長寬尺寸
    tar_Y, tar_X, _ = img_tar.shape
    # 創建目標影像的座標網格
    X, Y = np.meshgrid(np.arange(tar_X), np.arange(tar_Y))
    # 利用逆變換矩陣 H_inv 對座標點進行映射
    coords = np.dot(H_inv, np.stack((X.ravel(), Y.ravel(), np.ones(tar_X *
tar_Y))))
    # 將齊次座標轉換成笛卡爾座標，即將其除以第三個座標值
    coords /= coords[2]
    # 取出映射後的 x, y 座標
    coords = coords[:2, :]
    # 獲取原始影像 src 的長寬尺寸
    src_Y, src_X, _ = img_src.shape
    # 創建一個布林陣列 mask，用於標記映射後座標點是否落在原始影像的範圍內
    mask = (
        (coords[0, :] >= 0)
        & (coords[0, :] < src_X)
        & (coords[1, :] >= 0)
```

```python
        & (coords[1, :] < src_Y)
    ).reshape(tar_Y, tar_X)

    # 分別計算 x, y 的整數座標和小數部分
    x = coords[0, :].reshape(tar_Y, tar_X)
    y = coords[1, :].reshape(tar_Y, tar_X)
    x1 = np.floor(x).astype(int)
    y1 = np.floor(y).astype(int)
    # 確保座標不會超出原始影像的邊界
    x2 = np.minimum(x1 + 1, src_X - 1)
    y2 = np.minimum(y1 + 1, src_Y - 1)

    # 計算雙線性插值的權重
    dx = x - x1
    dy = y - y1
    w1 = (1 - dx) * (1 - dy)
    w2 = dx * (1 - dy)
    w3 = (1 - dx) * dy
    w4 = dx * dy

    # 從 mask 中篩選出有效的 y1, x1, y2, x2 座標
    valid_y1 = y1[mask]
    valid_x1 = x1[mask]
    valid_y2 = y2[mask]
    valid_x2 = x2[mask]

    # 利用雙線性插值根據權重和有效座標計算目標影像 tar 的像素值
    img_tar[mask] = (
        w1[mask][..., np.newaxis] * img_src[valid_y1, valid_x1]
        + w2[mask][..., np.newaxis] * img_src[valid_y1, valid_x2]
        + w3[mask][..., np.newaxis] * img_src[valid_y2, valid_x1]
        + w4[mask][..., np.newaxis] * img_src[valid_y2, valid_x2]
    )
```

```python
inverse_mapping(img_src, fig, H)
for p in corner_list:
    cv2.circle(fig, p, 5, (0, 0, 255), -1)
```


(c.)

```python
def compute_vanishing_point(corner_list):
    # 從角點列表中解包四個角點 A, B, C, D
    [A, B, C, D] = corner_list
    # 計算直線 AB 的座標表示的外積，得到直線的參數
    v1 = np.cross([A[0], A[1], 1], [B[0], B[1], 1])
    # 計算直線 CD 的座標表示的外積，得到直線的參數
    v2 = np.cross([C[0], C[1], 1], [D[0], D[1], 1])
    # 計算兩條直線的外積，得到兩直線的交點，即消失點
    vp = np.cross(v1, v2)
    # 將座標轉換成笛卡爾座標，即將其除以第三個座標值
    vp = vp / vp[2]

    # 返回消失點的整數座標 (x, y)
    return (int(vp[0]), int(vp[1]))
```

```python
# 計算消失點
vp = compute_vanishing_point(corner_list)
# 如果消失點的座標在目標圖像的範圍內
if (0 <= vp[0] < tar_X) & (0 <= vp[1] < tar_Y):
    # 在圖像上繪製一個紅色的圓來標示消失點
    cv2.circle(fig, vp, 5, (0, 0, 255), -1)
```