

影像處理作業 2

- 系級：資訊工程系一年級
- 姓名：楊啟弘
- 學號：7113056083

本作業是一個全景（Panorama）影像拼接工具，透過模組化設計將流程拆分為：讀圖、特徵偵測、特徵匹配、單應性估計、影像投影與融合等步驟。整體架構與運作流程如下：

一、專案目標、要求

輸入：多張重疊且視角略有差異的影像（需三張以上） 輸出：一張無縫接合的全景影像 要求：

- Affine transform 不可依靠套件 (`transformer.py`)

二、主要程式檔與模組職責

此專案將核心功能拆分為多個模組，各模組職責如下：

檔案	功能說明
<code>config/settings.yaml</code>	拼接參數設定（特徵偵測器、匹配器、RANSAC、混合方式等）
<code>loader.py</code>	影像讀取與前處理（批次載入、灰階轉換、調整大小）
<code>feature.py</code>	特徵偵測與描述子計算（ORB、SIFT、AKAZE 等）
<code>matcher.py</code>	描述子匹配邏輯（BFMatcher、FlannBasedMatcher 與 Lowe's ratio test）
<code>transformer.py</code>	仿射矩陣估算（RANSAC）、影像仿射變形與矩陣合成
<code>blender.py</code>	影像融合方法（羽化混合 feather、金字塔多頻帶 multiband）
<code>stitcher.py</code>	主拼接流程整合：讀設定、載入影像、估算變換、Warp、混合、輸出

三、工作流程

以下是在每個流程步驟中，插入專案對應程式碼的詳細說明：

1. 影像讀取與前處理 (`loader.py`)

```
from loader import load_images_from_dir, preprocess_image

# 批次載入 data/input 資料夾
images_dict = load_images_from_dir('data/input/')
# 若需要灰階或縮放
gray = preprocess_image(img, to_gray=True, resize=(800, 600))
```

- `load_images_from_dir` 會回傳 {'img1.jpg': ndarray, ...}

- preprocess_image 支援灰階轉換與鎖定尺寸

2. 影像讀取與前處理 (`loader.py`)

```
from feature import get_feature_detector, detect_and_compute

# 建立 ORB 偵測器
detector = get_feature_detector('ORB', nfeatures=2000)
# 計算關鍵點與描述子
kp, des = detect_and_compute(detector, gray_image)
```

3. 描述子匹配 (`matcher.py`)

```
from matcher import create_matcher, match_descriptors

# 建立 BFMatcher
matcher = create_matcher('BF', norm_type=cv2.NORM_HAMMING,
cross_check=False)
# KNN + Lowe's ratio test，取前 50 筆
matches = match_descriptors(
    matcher, des1, des2,
    ratio_test=True, ratio=0.75, top_k=50
)
```

4. 仿射矩陣估算與串接 (`transformer.py`)

```
from transformer import estimate_affine_transform, compose_transforms

# src_pts, dst_pts shape 都為 (N,1,2)
M23, inlier_mask = estimate_affine_transform(
    src_pts, dst_pts,
    ransac_thresh=5.0, max_iters=2000
)
# 若要多段串接：
M3x3 = np.vstack([M23, [0, 0, 1]])
T_total = compose_transforms([M3x3, prev_T])
```

5. 畫布大小計算

```
# 計算所有影像投影後的四角
all_corners = []
for img, T in zip(imgs, transforms):
    h, w = img.shape[:2]
    corners = np.array([[0, 0, 1], [w, 0, 1], [w, h, 1], [0, h, 1]]).T
    proj = T @ corners
    proj = proj[:2] / proj[2]
```

```
    all_corners.append(proj.T)
all_pts = np.vstack(all_corners)
min_x, min_y = np.min(all_pts, axis=0)
max_x, max_y = np.max(all_pts, axis=0)
```

6. 影像 Warp ([transformer.py](#))

```
from transformer import warp_image

# 對每張影像貼到畫布
warp = warp_image(
    img,          # 原圖
    M[:2],        # 2x3 仿射矩陣
    (out_w, out_h),
    flags=cv2.INTER_LINEAR,
    borderMode=cv2.BORDER_CONSTANT,
    borderValue=0
)
# 同時計算 mask
mask = warp_image(
    np.ones((h,w),dtype=np.uint8)*255,
    M[:2], (out_w, out_h)
)
```

7. 影像融合 ([blender.py](#))

```
from blender import blend_images

# 第一張直接貼上
panorama = warp_images[0]
# 後續逐張混合
panorama = blend_images(
    panorama, warped, mask,
    method='feather', blur_radius=21
)
```

8. 整合驅動 ([stitcher.py](#))

```
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--config', required=True)
    parser.add_argument('--input', required=True)
    parser.add_argument('--output', required=True)
    args = parser.parse_args()
    stitch_images(args.config, args.input, args.output)
```

- stitch_images 依序呼叫上面各模組，處理錯誤檢查並輸出結果

如此，每個演算法步驟都對應到專案中具體的程式碼片段，方便閱讀與維護！

四、成果範例展示

以下展示輸入影像與拼接後輸出結果示例，

影像 1



影像 2



影像 3



- 輸出全景圖：data/output/panorama_example.jpg



五、討論

多虧於現今LLM的技術使這份作業簡單得許多，如果沒有LLM肯定要花上數倍的時間，甚至無法完成。但即使是使用了LLM，無法使用cv2來實作affine transform也讓我額外花了更多時間開發影像拼接。這份作業讓我更加了解影像拼接的流程，以及還能夠使用哪些演算法來實作出來。

六、Affine Transform

```
def estimate_affine_ransac(src_pts, dst_pts, ransac_thresh=5.0,
max_iters=2000):
    """
    使用 RANSAC 從 src_pts → dst_pts 估算 2x3 仿射矩陣。
    回傳:
        M : (2,3) 仿射矩陣
        mask: (N,) 內點標記 1/0
    """
    N = src_pts.shape[0]
    if N < 3:
        raise ValueError("至少需要 3 對點才能估算仿射變換。")

    best_M = None
    best_inliers = np.zeros(N, dtype=bool)
    best_count = 0

    # 將點加上常數項，方便一次求解
    def solve_affine(p_src, p_dst):
        # p_src, p_dst shape=(k,2), k>=3
        # 方程: [ x y 1 0 0 0 ] [a]   [x']
        #           [ 0 0 0 x y 1 ] [b] = [y']
        A = []
        b = []
        for (x, y), (xp, yp) in zip(p_src, p_dst):
            A.append([x, y, 1, 0, 0, 0])
            A.append([0, 0, 0, x, y, 1])
            b.append(xp)
            b.append(yp)
        A = np.array(A)      # shape=(2k,6)
        b = np.array(b)      # shape=(2k,)
        # 最小二乘解
        x, *_ = np.linalg.lstsq(A, b, rcond=None)
        # x = [a, b, tx, c, d, ty]
        return np.array([[x[0], x[1], x[2]],
                       [x[3], x[4], x[5]]])

    for i in range(max_iters):
        # 隨機抽 3 對點
        idx = np.random.choice(N, size=3, replace=False)
        M_candidate = solve_affine(src_pts[idx], dst_pts[idx])

        # 計算所有點的重投影誤差
        src_h = np.hstack([src_pts, np.ones((N,1))])          # shape=(N,3)
        proj = (M_candidate @ src_h.T).T                      # shape=(N,2)
        errs = np.linalg.norm(proj - dst_pts, axis=1)         # shape=(N,)
```

```
# 以閾值分類內點
inliers = errs < ransac_thresh
count = inliers.sum()

if count > best_count:
    best_count = count
    best_inliers = inliers
    best_M = M_candidate

# 若已經大部分皆為內點，就可提早停止
if count > 0.8 * N:
    break

if best_M is None:
    raise RuntimeError("RANSAC 無法估算出有效模型。")

# 用所有內點再做一次最小二乘擬合以精緻化矩陣
best_M = solve_affine(src_pts[best_inliers], dst_pts[best_inliers])

# 回傳 M 與內點遮罩 (1/0)
return best_M, best_inliers.astype(np.uint8)
```

def estimate_affine_transform(kp1, kp2, matches, ransac_thresh=5.0, max_iters=2000):
 ...

根據匹配的關鍵點估算仿射變換矩陣。

參數：

- kp1 (list of cv2.KeyPoint): 參考影像的關鍵點
- kp2 (list of cv2.KeyPoint): 待變換影像的關鍵點
- matches (list of cv2.DMatch): 兩影像之間的匹配列表
- ransac_thresh (float): RANSAC 重投影閾值，預設 5.0
- max_iters (int): RANSAC 最大迭代次數，預設 2000

回傳：

- M (ndarray of shape (2,3)): 估算出的仿射矩陣
- mask (ndarray): 表示內點(inliers)的遮罩

....

構造點陣

```
src_pts = np.float32([kp2[m.trainIdx].pt for m in
matches]).reshape(-1, 1, 2)
dst_pts = np.float32([kp1[m.queryIdx].pt for m in
matches]).reshape(-1, 1, 2)
```

....

使用 RANSAC 估算仿射矩陣

```
M, mask = cv2.estimateAffinePartial2D(
    src_pts,
    dst_pts,
    method=cv2.RANSAC,
    ransacReprojThreshold=ransac_thresh,
    maxIters=max_iters
)
```

```
    ...  
  
    src = src_pts.reshape(-1, 2)  
    dst = dst_pts.reshape(-1, 2)  
    M, mask = estimate_affine_ransac(src, dst,  
                                      ransac_thresh=ransac_thresh,  
                                      max_iters=max_iters)  
  
    return M, mask
```