МІНІСТЕРСТВО ОСВІТИ І НАУКИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Кафедра систем штучного інтелекту

Лабораторна робота №2

З курсу “Обробка зображень методами штучного інтелекту”

Виконав:
студент групи КН-408
Марій Павло

Викладач:
Пелешко Д. Д.

Львів – 2022

Тема: Суміщення зображень на основі використання дескрипторів.

Мета: Навчитись вирішувати задачу суміщення зображень засобом видобування особливих точок і використання їх в процедурах матчінгу.

Теоретичні відомості

Метод SIFT.

У 2004 році Д.Лоу, Університет Британської Колумбії, придумав алгоритм - Scale Invariant Feature Transform (SIFT), який видобуває ключові (особливі) точки і обчислює їх дескриптори.

Загалом алгоритм SIFT складається з п’яти основних етапів:

1. Виявлення масштабно-просторових екстремумів (Scale-space Extrema Detection) - основним завданням етапу є виділення локальних екстремальних точок засобом побудови пірамід гаусіанів (Gaussian) і різниць гаусіанів (Difference of Gaussian, DoG).

2. Локалізація ключових точок (Keypoint Localization) - основним завданням етапу є подальше уточнення локальних екстремумів з метою фільтрації їх набору - тобто видалення з подальшого аналізу точок, які є краєвими, або мають низьку контрастність.

3. Визначення орієнтації (Orientation Assignment) - для досягнення інваріантності повороту растра на цьому етапі кожній ключовій точці присвоюється орієнтація.

4. Дескриптор ключових точок (Keypoint Descriptor) - завданням етапу є побудова дескрипторів, які містять інформацію про окіл особливої точки для задачі подальшого порівння на збіг.

5. Зіставлення по ключових точках (Keypoint Matching) - пошук збігів для вирішення завдання суміщення зображень.

Алгоритм RANSAC - Random sample consensus

Для досягнення високої точності визначення збігів об'єктів на зображеннях зазвичай відфільтрувати дескриптори тільки за відстанню є недостатньо. Якщо об'єкт рухається на сцені або зображений з іншого ракурсу, то при застосуванні трансформації «накладення» n точок одного зображення на відповідні по найближчому сусіду n точок іншого, можна виявити особливості, що не відносяться до загального об'єкту і тим самим зменшити кількість хибно виявлених зв'язків.

Схема роботи алгоритму RANSAC полягає в циклічному повторенні пошуку матриці трансформації HH між чотирма особливими точками s_i, які

випадково обираються i на одному зображенні, і відповідними їм точками на другому:

$$s_i \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \sim H \begin{bmatrix} x'_i \\ y'_i \\ 1 \end{bmatrix}$$

Найкращою матрицею трансформації вважається та, в якій досягнуто мінімум суми відхилень будь-яких спеціальних точок зображень при перетворенні НH, за задану кількість циклів (≤ 2000):

$$\sum_i \left[(x_i - \frac{h_{11}x'_i + h_{12}y'_i + h_{13}}{h_{31}x'_i + h_{32}y'_i + h_{33}})^2 + (y_i - \frac{h_{21}x'_i + h_{22}y'_i + h_{23}}{h_{31}x'_i + h_{32}y'_i + h_{33}})^2 \right]$$

У підсумкову множину srcPoints’ додаються тільки ті точки $srcPo$ srcPointsi$i$, відхилення яких становить менше заданого порогу:

$$|dstPointsi - H * srcPointsi| < reprojThreshold$$

де srcPoints$srcPoin$ - множина усіх особливих точок першого зображення, а dstPoints - множина відповідних їм особливих точок другого.

Хід роботи

Варіант – 10. Номер в списку групи – 21.

Завдання:

Вибрати з інтернету набори зображень з різною контрастністю і різним флуктуаціями освітленості. Для кожного зображення побудувати варіант спотвореного (видозміненого зображення). Для кожної отриманої пари побудувати дескриптор і проаналізувати можливість суміщення цих зображень і з визначення параметрів геметричних перетворень (кут повороту, зміщень в напрямку x і напрямку y).

BRIEF.

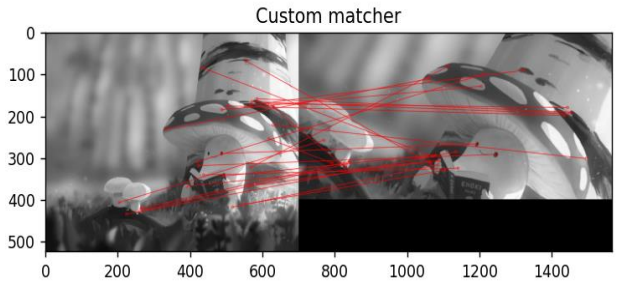
Для перевірки збігів необхідно написати власну функцію матчінгу, а результати її роботи перевірити засобами OpenCV. Якщо повної реалізації дескриптора не має в OpenCV, то такий необхідно створити власну функцію побудови цих дискрипторів. У цьому випадку матчінг можна здійснювати стандартними засобами (якщо це можливо).

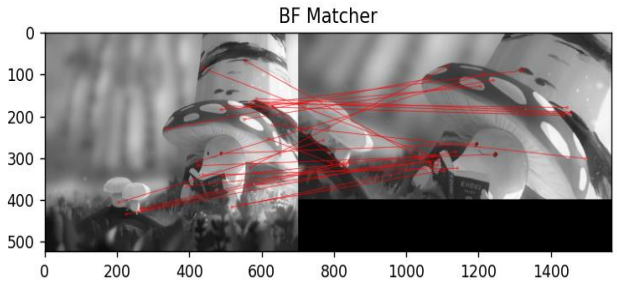
Код програми:

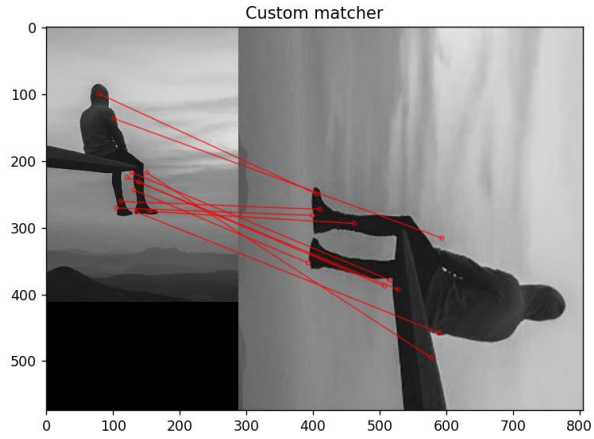
```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt

def match(des1, des2, ratio=0.98):
    match1=[]
    match2=[]
    distances = {}
    for i in range(des1.shape[0]):
        if np.std(des1[i,:])!=0:
            d = np.zeros(des2.shape[0])
            for j in range(des2.shape[0]):
                d[j]=cv.norm(des1[i, :], des2[j, :], cv.NORM_HAMMING)
            orders =np.argsort(d).tolist()
            if d[orders[0]]/d[orders[1]]<=ratio:
                match1.append((i,orders[0]))
            distances[f'{i}-{orders[0]}'] = d[orders[0]]

    for i in range(des2.shape[0]):
        if np.std(des2[i,:])!=0:
            d = np.zeros(des1.shape[0])
            for j in range(des1.shape[0]):
                d[j]=cv.norm(des2[i, :], des1[j, :], cv.NORM_HAMMING)
            orders =np.argsort(d).tolist()
            if d[orders[0]]/d[orders[1]]<=ratio:
                match2.append((orders[0],i))
            distances[f'{orders[0]}-{i}'] = d[orders[0]]

    ##find good matches in rotation tests both ways
    match = list(set(match1).intersection(set(match2)))
    return [(pair[0], pair[1], distances[f'{pair[0]}-{pair[1]}']) for pair in
match]

images = []
images.append( [cv.imread('./image1.jpg', cv.IMREAD_GRAYSCALE),
cv.imread('./image2.jpg', cv.IMREAD_GRAYSCALE)])
images.append( [cv.imread('./image3.jpg', cv.IMREAD_GRAYSCALE),
cv.imread('./image4.jpg', cv.IMREAD_GRAYSCALE)])

for i in images:
    image1, image2 = i

    star = cv.xfeatures2d.StarDetector_create()
    brief = cv.xfeatures2d.BriefDescriptorExtractor_create()

    temp1 = star.detect(image1, None)
    temp2 = star.detect(image2, None)

    image1_keypoints, image1_descriptor = brief.compute(image1, temp1)
```

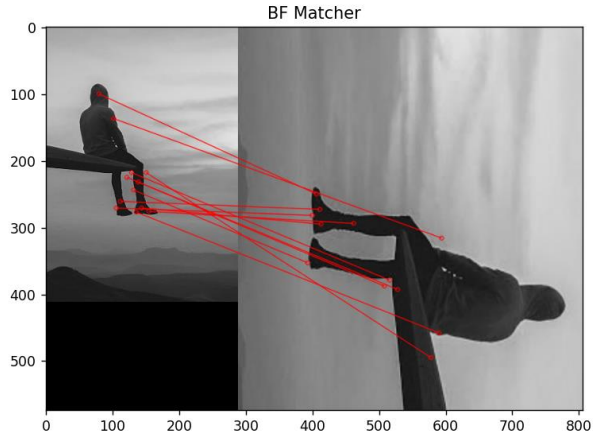
```
    image2_keypoints, image2_descriptor = brief.compute(image2, temp2)

    matches = match(image1_descriptor, image2_descriptor)
    matches = sorted([cv.DMatch(*i) for i in matches], key=lambda x: x.distance)

    bf = cv.BFMatcher(cv.NORM_HAMMING, crossCheck=True)
    matches_bf = bf.match(image1_descriptor, image2_descriptor)
    matches_bf = sorted(matches_bf, key = lambda x:x.distance)

    fig, axis = plt.subplots(1, 2)
    img3 = cv.drawMatches(image1, image1_keypoints, image2, image2_keypoints,
matches[:30], None, **dict(
        matchColor = (255, 0, 0),
        singlePointColor = (255,0,0),
        flags = cv.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS
    ))
    img3_bf = cv.drawMatches(image1, image1_keypoints, image2, image2_keypoints,
matches_bf[:30], None, **dict(
        matchColor = (255, 0, 0),
        singlePointColor = (255,0,0),
        flags = cv.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS
    ))

    axis[0].imshow(img3)
    axis[0].set_title("Custom matcher")
    axis[1].imshow(img3_bf)
    axis[1].set_title("BF Matcher")
    plt.show()
```

Результати роботи програми:









Коментарі: Спершу я зчитую зображення в чорно-білих кольорах. Для визначення особливих точок я використав Star Detection, він має кращі обчислювальні характеристики. Дескриптори обчислював вбудованим в opencv Brief Descriptor. Після цього провів процедуру матчингу двома способами – власним матчером та Brute Force матчером. В кінці я показую результати матчингів на зображеннях, порівнюючи їх.

Висновки: Я навчився вирішувати задачу суміщення зображень засобом видобування особливих точок і використав їх в процедурах матчінгу.

Порівнюючи результати роботи вбудованого матчера та власного, можна сказати, що працюють вони доволі схоже. Зображення, які ми отримали в результаті, дуже схожі між собою, матчінг майже однаковий, проте є деякі відмінності. Власний матчер приймає параметр ratio, який задає «прискіпливість» алгоритму. В процесі матчингу було застосовано норму Хемінга, як і було рекомендовано, а також було реалізовано кросматчінг, який спершу обраховує матчі в двох напрямках – forward та backward, а потім вибирає лише ті, які співпадають при обрахунках.