Лабораторна робота № 2.

Суміщення зображень на основі використання дескрипторів

Мета - навчитись вирішувати задачу суміщення зображень засобом видобування особливих точок і використання їх в процедурах матчінгу

1. Короткі теоретичні відомості

1.1. Метод SIFT (Scale-Invariant Feature Transform)

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 алгоритмі 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 побудова пірамід гаусіанів (Gaussian) і різниць гаусіанів (Difference of Gaussian, DoG).

2. Локалізація ключових точок (Keypoint Localization) - основним завданням етапу є подальше учточнення 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 етпау є побудова дескрипторів, які містять інформація про окіл особливої точки для задачі подальшого порівння на збіг.

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

1.2. Алгоритм RANSAC - Random sample consensus

Для досягнення високої точності визначення збігів об'єктів на зображеннях зазвичай відфільтрувати дескриптори тільки за відстані 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:

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

Нйкращою матрицею трансформації вважається та, в якій досягнуто мінімум суми відхилень будь-яких спеціальних точок зображень при перетворенні HH, 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$srcPo$ $srcPointsi$, відхилення яких становить менше заданого порогу:

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

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

Детально усі необхідні теоретичні відомості можна знайти у презентації до лекції №3 або в інтернеті.

2. SIFT в OpenCV

Визначенням особливих точок і їх відображення засобами OpenCV є таким

```
import numpy as np
import cv2 as cv
img = cv.imread('home.jpg')
gray= cv.cvtColor(img,cv.COLOR_BGR2GRAY)
sift = cv.SIFT_create()
kp = sift.detect(gray,None)
img=cv.drawKeypoints(gray,kp,img)
```

```
cv.imwrite('sift_keypoints.jpg',img)
```

Функція sift.detect () знаходить ключову точку на зображенні. Їй параметром можна передати маску, якщо зона пошуку є обмежена. Кожна ключова точка представляє собою спеціальну структуру, яка містить множину таких атрибутів: координати, розмір околу, напрямок та ін.

У OpenCV також передбачена функція cv.drawKeyPoints (), яка за розміщенням ключових точек малює невеликі окружності . Параметр cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS, визначає режим рисування особливих точок із вказанням орієнтації

```
img=cv.drawKeypoints(gray,kp,img,flags=cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
cv.imwrite('sift_keypoints.jpg',img)
```

Наприклад

Для обчислення SIFT дескриптора в OpenCV пропонується два методи.

1. Якщо ключові точки вже знайдені, то функція sift.compute () по них обчислить дескриптори. Наприклад: kp, des = sift.compute (gray,kp)

2. Якщо ключові точки попередньо не знайдені, то для визначення дескрипторів треба використовувати функцію sift.detectAndCompute ().

Другий спосіб можна записати як

```
sift = cv.SIFT_create()
kp, des = sift.detectAndCompute(gray,None)
```

Результат повернення kp буде списком ключових точок, des - numpy array з розмірністю $\text{Number\_of\_Keypoints} \times 128$.

3. Суміщення зображень засобом Feature Matching

3.1. Основи Brute-Force Matcher

Brute-Force matcher (BF-matcher) є реалізовує простий матчінг метод. Він приймає дескриптор однієї ознаки в першій множині і зіставляє за деякою метрикою з усіма ознаками в другій множині. Як результат повертається найближчий дексриптор (ознака).

Для використання BF-matcher спочатку використовуючи функцію cv.BFMatcher () необхідно створити об'єкт BFMatcher. Функція має два необов'язкові параметри. Перший - normType. Він задає тип метрики для вимірювання відстані між дескрипторами. За замовчуванням використовується метрика L2 (cv.NORM_L2). Для SIFT та SURF і т.д. також рекомендується використовувати метрику L1 (cv.NORM_L1).

Для дескрипторів, заснованих на бінарних рядках, таких як ORB, BRIEF, BRISK і т.д., треба використовувати метрику Хемінга (cv.NORM_HAMMING). Якщо ORB використовує WTA_K == 3 або 4, то слід використовувати cv.NORM_HAMMING2.

Другий параметр функції створення матчера є булівська змінна crossCheck, яка за замовчуванням рівна False. Якщо встановити її в True, то Matcher повертає тільки ті збіги, коли обидві ознаки в обох множинах збігаються одна з одною.

Після його створення матчера його двома важливими методами є BFMatcher.match () і BFMatcher.knnMatch (). Перший повертає кращий збіг. Другий метод - повертає k кращих збігів, де k задається параметром.

Як і ми використовували cv.drawKeypoints () для відтворення ключових точок, cv.drawMatches () допомагає нам малювати відповідності. Вона складає два зображення по горизонталі і малює лінії від першого зображення до другого, показуючи найкращі збіги. Також існує cv.drawMatchesKnn, який відображає всі k кращих збігів. Якщо k = 2, то він відобразить дві лінії збігів для кожної ключової точки. Тому ми повинні передати маску, якщо ми хочемо вибірково намалювати її.

3.2. Використання Brute-Force Matching з ORB Descriptors

Наведемо простий приклад того, як зіставляти функції між двома зображеннями. Вхідними зображеннями є queryImage і trainImage. Основне завдання - знайти queryImage в trainImage, використовуючи можливості порівняння.

Використовуємо дескриптори ORB для співставлення ознак. Напочатку завантажуємо зображення і визначаємо дескриптори.

```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
img1 = cv.imread('box.png',cv.IMREAD_GRAYSCALE) # queryImage
img2 = cv.imread('box_in_scene.png',cv.IMREAD_GRAYSCALE) # trainImage
# Initiate ORB detector
orb = cv.ORB_create()
# find the keypoints and descriptors with ORB
kp1, des1 = orb.detectAndCompute(img1,None)
kp2, des2 = orb.detectAndCompute(img2,None)
```

Створимо об’єкт BFMatcher з метрикою cv.NORM_HAMMING, оскільки будуть використовуватись дескриптори ORB, та із параметром crossCheck. Метод Matcher.match() використовуватиметься для отримання кращі збігів в двох зображеннях.

Для зручності кращі збіги сортуємо в порядку зростання їх відстаней так, щоб кращі збіги були першими. А далі можна відобразити перші 10 збігів візуально.

```
    # create BFMatcher object
    bf = cv.BFMatcher(cv.NORM_HAMMING, crossCheck=True)
    # Match descriptors.
    matches = bf.match(des1,des2)
    # Sort them in the order of their distance.
    matches = sorted(matches, key = lambda x:x.distance)
    # Draw first 10 matches.
    img3 =
cv.drawMatches(img1,kp1,img2,kp2,matches[:10],None,flags=cv.DrawMatchesFlags_NOT
_DRAW_SINGLE_POINTS)
    plt.imshow(img3),plt.show()
```

Приклад виконання наведеного коду зображено на рисунку

3.3. Що таке Matcher Object?

Результатом виклику функції Matcher.match() є список DMatch об'єктів. Кожен DMatch об'єкт має такі атрибути:

- DMatch.distance - відстань між дескрипторами.
- DMatch.trainIdx - ідекс дескриптора у списку train descriptors
- DMatch.queryIdx - ідекс дескриптора у списку query descriptors
- DMatch.imgIdx - індекс train image.

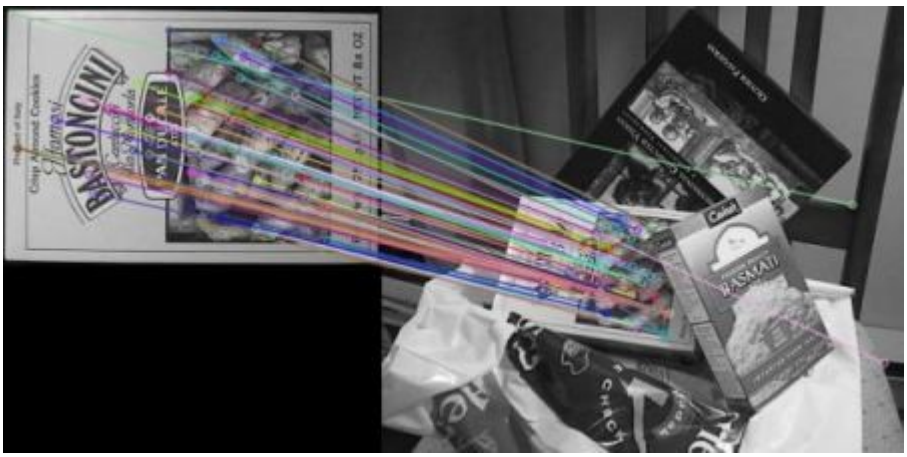
3.4. Brute-Force Matching з SIFT дескрипторами Ratio Test

Для цього треба використовувати метод Matcher.knnMatch(), наприклад

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

```
    img1 = cv.imread('box.png',cv.IMREAD_GRAYSCALE) # queryImage
    img2 = cv.imread('box_in_scene.png',cv.IMREAD_GRAYSCALE) # trainImage
    # Initiate SIFT detector
    sift = cv.SIFT_create()
    # find the keypoints and descriptors with SIFT
    kp1, des1 = sift.detectAndCompute(img1,None)
    kp2, des2 = sift.detectAndCompute(img2,None)
    # BFMatcher with default params
    bf = cv.BFMatcher()
    matches = bf.knnMatch(des1,des2,k=2)
    # Apply ratio test
    good = []
    for m,n in matches:
    if m.distance < 0.75*n.distance:
    good.append([m])
    # cv.drawMatchesKnn expects list of lists as matches.
    img3 =
cv.drawMatchesKnn(img1,kp1,img2,kp2,good,None,flags=cv.DrawMatchesFlags_NOT_DRAW
_SINGLE_POINTS)
    plt.imshow(img3),plt.show()
```

Приклад виконання наведеного коду зображено на рисунку



3.5. Matcher на основі FLANN

FLANN (Fast Library for Approximate Nearest Neighbors). Він містить набір алгоритмів, оптимізованих для швидкого пошуку найближчих сусідів у великих наборах даних і для фічей з великою розмірністю. Він працює швидше, ніж BFMatcher для великих наборів даних. Ми розглянемо другий приклад з матчером на основі FLANN.

Для матчера на основі FLANN нам потрібно передати два словника з параметри, які визначають використовуваний алгоритм. Перший - IndexParams. Для різних алгоритмів передана інформація пояснюється в документах FLANN. Резюмуючи, для таких алгоритмів, як SIFT, SURF і т.д., можна передати наступне:

```
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
```

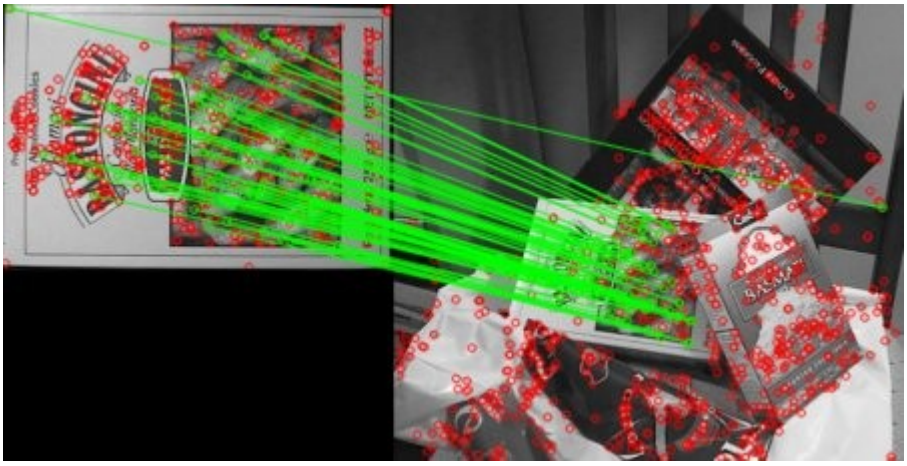
Під час використання ORB, ви можете передати наступне. Коментовані значення рекомендуються в статтях, але в деяких випадках вони не дають бажаних результатів. Інші значення працювали нормально:

```
FLANN_INDEX_LSH = 6
index_params= dict(algorithm = FLANN_INDEX_LSH,
table_number = 6, # 12
key_size = 12, # 20
multi_probe_level = 1) #2
```

Другий словник - SearchParams. Він задає кількість разів, коли дерева в індексі повинні бути рекурсивно оброблені. Більш високі значення дають більшу точність, але також вимагають більше часу. Якщо ви хочете змінити значення, передайте search_params = dict (check = 100).

```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
img1 = cv.imread('box.png',cv.IMREAD_GRAYSCALE) # queryImage
img2 = cv.imread('box_in_scene.png',cv.IMREAD_GRAYSCALE) # trainImage
# Initiate SIFT detector
sift = cv.SIFT_create()
# find the keypoints and descriptors with SIFT
kp1, des1 = sift.detectAndCompute(img1,None)
kp2, des2 = sift.detectAndCompute(img2,None)
# FLANN parameters
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks=50) # or pass empty dictionary
flann = cv.FlannBasedMatcher(index_params,search_params)
matches = flann.knnMatch(des1,des2,k=2)
# Need to draw only good matches, so create a mask
matchesMask = [[0,0] for i in range(len(matches))]
# ratio test as per Lowe's paper
for i,(m,n) in enumerate(matches):
if m.distance < 0.7*n.distance:
matchesMask[i]=[1,0]
draw_params = dict(matchColor = (0,255,0),
singlePointColor = (255,0,0),
matchesMask = matchesMask,
flags = cv.DrawMatchesFlags_DEFAULT)
img3 = cv.drawMatchesKnn(img1,kp1,img2,kp2,matches,None,**draw_params)
plt.imshow(img3,),plt.show()
```

Результати наведено нижче



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).

1. SIFT
2. PCA-SIFT
3. GLOH
4. DAISY
5. A-KAZE
6. SURF
7. FAST,
8. BRISK
9. LDB
10. BRIEF,
11. ORB

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).

Додаткове завдання (оцінюється у додаткові бали) - в процесі порівняння дескрипторів використати власну реалізацію алгоритму RANSAC.
Додаткові бали також надаються за написання власної функції побудови дексрипторів, без використання Ope.CV

Додаткова література:

http://www.edwardrosten.com/work/rosten_2005_tracking.pdf
http://www.edwardrosten.com/work/rosten_2006_machine.pdf
https://arxiv.org/pdf/0810.2434.pdf
https://www.cs.ubc.ca/~lowe/525/papers/calonder_eccv10.pdf
http://www.willowgarage.com/sites/default/files/orb_final.pdf