

Thành viên:

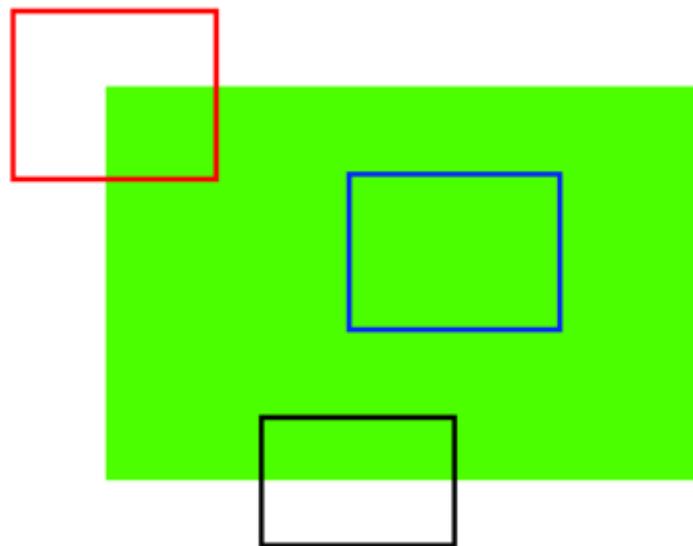
Nguyễn Văn Tuấn - B20DCPT182

Nguyễn Quốc Đạt - B20DCPT053

Nguyễn Hồng Nhung - B20DCPT150

1) What is called features of objects in an image? give some examples those called features?

- Features of objects in an image (Đặc trưng của đối tượng trong ảnh): Đề cập tới những đặc điểm hay các mẫu đặc biệt dễ theo dõi và dễ so sánh, được sử dụng cho các nhiệm vụ khác nhau như nhận dạng đối tượng, phân tích hình ảnh, phân loại hình ảnh ...
- Ví dụ:



Miếng màu xanh lam là vùng bằng phẳng, khó tìm và theo dõi. Bất cứ nơi nào khi di chuyển miếng màu xanh, nó trông giống nhau. Miếng màu đen có một cạnh. Nếu di chuyển nó

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
MIN_MATCH_COUNT = 10
img1 = cv.imread('img1.png', cv.IMREAD_GRAYSCALE)
img2 = cv.imread('img2.png', cv.IMREAD_GRAYSCALE)
sift = cv.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks = 50)
flann = cv.FlannBasedMatcher(index_params, search_params)
matches = flann.knnMatch(des1,des2,k=2)
# store all the good matches as per Lowe's ratio test.
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append(m)
if len(good)>MIN_MATCH_COUNT:
    src_pts = np.float32([ kp1[m.queryIdx].pt for m in good ]).reshape(-1,1,2)
    dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good ]).reshape(-1,1,2)
    M, mask = cv.findHomography(src_pts, dst_pts, cv.RANSAC,5.0)
    matchesMask = mask.ravel().tolist()
```

theo chiều dọc (tức là dọc theo độ dốc) thì nó sẽ thay đổi. Nhưng nếu di chuyển theo chiều ngang (song song với cạnh) thì nó trông giống nhau. Và đối với miếng màu đỏ, đó là góc. Bất cứ nơi nào di chuyển nó, nó trông khác nhau, có nghĩa là nó là duy nhất. Vì vậy, về cơ bản, các góc được coi là đặc điểm tốt của một bức ảnh. (Không chỉ các góc, trong một số trường hợp, các đốm màu được coi là đặc điểm tốt).

2) Function in openCV use for Harris corner detection? A snipcode demo?

Về mặt lý thuyết:

$$E(u, v) = \sum_{x,y} \underbrace{w(x, y)}_{\text{window function}} \underbrace{[I(x + u, y + v) - I(x, y)]^2}_{\text{shifted intensity}} / \underbrace{I(x, y)}_{\text{intensity}}$$

$$E(u, v) \approx [u \quad v] M \begin{bmatrix} u \\ v \end{bmatrix}$$

$$M = \sum_{x,y} w(x, y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix}$$

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
MIN_MATCH_COUNT = 10
img1 = cv.imread('img1.png', cv.IMREAD_GRAYSCALE)
img2 = cv.imread('img2.png', cv.IMREAD_GRAYSCALE)
sift = cv.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks = 50)
flann = cv.FlannBasedMatcher(index_params, search_params)
matches = flann.knnMatch(des1,des2,k=2)
# store all the good matches as per Lowe's ratio test.
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append(m)
if len(good)>MIN_MATCH_COUNT:
    src_pts = np.float32([ kp1[m.queryIdx].pt for m in good ]).reshape(-1,1,2)
    dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good ]).reshape(-1,1,2)
    M, mask = cv.findHomography(src_pts, dst_pts, cv.RANSAC,5.0)
    matchesMask = mask.ravel().tolist()
```

$$R = \det(M) - k(\text{trace}(M))^2$$

w: ma trận cửa sổ.

l: ma trận cường độ

lambda1, lamb2a là giá trị riêng của ma trận M

$\det(M) = \lambda_1 * \lambda_2$

$\text{trace}(M) = \lambda_1 + \lambda_2$

R:

Khi $|R|$ nhỏ, điều này xảy ra khi λ_1 và λ_2 nhỏ, vùng bằng phẳng.

Khi $R < 0$, điều đó xảy ra khi $\lambda_1 > > \lambda_2$ hoặc ngược lại, vùng đó là cạnh.

Khi R lớn, điều này xảy ra khi λ_1 và λ_2 lớn và $\lambda_1 \sim \lambda_2$, vùng này là một góc.

OpenCV có hàm [cv.cornerHarris\(\)](#) cho mục đích này. Các tham số cụ thể là:

- img - Hình ảnh đầu vào. Nó phải là ảnh xám và kiểu dữ liệu float32.
- blockSize - Kích thước của vùng lân cận được xem xét.
- ksize - Tham số khâu độ của đạo hàm Sobel.
- k - Tham số tự do máy dò Harris.

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
MIN_MATCH_COUNT = 10
img1 = cv.imread('img1.png', cv.IMREAD_GRAYSCALE)
img2 = cv.imread('img2.png', cv.IMREAD_GRAYSCALE)
sift = cv.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks = 50)
flann = cv.FlannBasedMatcher(index_params, search_params)
matches = flann.knnMatch(des1,des2,k=2)
# store all the good matches as per Lowe's ratio test.
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append(m)
if len(good)>MIN_MATCH_COUNT:
    src_pts = np.float32([ kp1[m.queryIdx].pt for m in good ]).reshape(-1,1,2)
    dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good ]).reshape(-1,1,2)
    M, mask = cv.findHomography(src_pts, dst_pts, cv.RANSAC,5.0)
    matchesMask = mask.ravel().tolist()
```

```

1 import numpy as np
2 import cv2 as cv
3
4 img = cv.imread('chess.png')
5 gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
6 gray = np.float32(gray)
7
8 dst = cv.cornerHarris(gray, 2, 3, 0.04)
9 dst = cv.dilate(dst, None)
10 img[dst > 0.01*dst.max()] = [0,0,255]
11 cv.imshow('dst',img)
12 if cv.waitKey(0) & 0xff == 27:
13     cv.destroyAllWindows()

```

The screenshot shows a Python code editor with a file named 'harry_corner_dectector.py'. The code uses OpenCV to detect corners in a chessboard image ('chess.png'). It applies the Harris corner detection method with parameters (2, 3, 0.04), dilates the result, and then overlays red corners onto the original image. A window titled 'dst' displays the processed chessboard with red corner markers at each intersection.

3) What is cons of Harris corner? What is Shi-Tomasi Corner Detector ? Are they Good Features to Track? A snipcode demo?

Với Harris corner:

$$R = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2$$

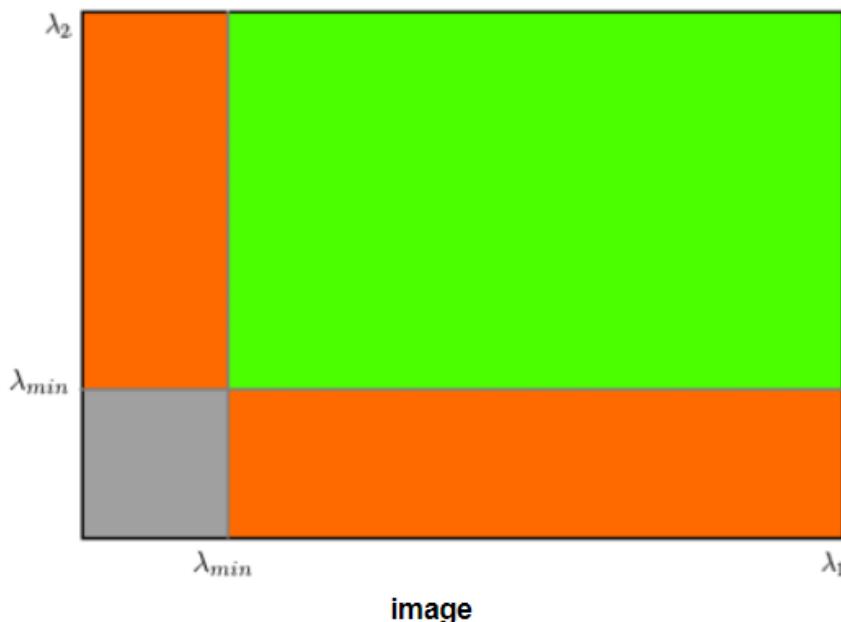
$$R = \min(\lambda_1, \lambda_2)$$

```

import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
MIN_MATCH_COUNT = 10
img1 = cv.imread('img1.png', cv.IMREAD_GRAYSCALE)
img2 = cv.imread('img2.png', cv.IMREAD_GRAYSCALE)
sift = cv.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks = 50)
flann = cv.FlannBasedMatcher(index_params, search_params)
matches = flann.knnMatch(des1,des2,k=2)
# store all the good matches as per Lowe's ratio test.
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append(m)
if len(good)>MIN_MATCH_COUNT:
    src_pts = np.float32([ kp1[m.queryIdx].pt for m in good ]).reshape(-1,1,2)
    dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good ]).reshape(-1,1,2)
    M, mask = cv.findHomography(src_pts, dst_pts, cv.RANSAC,5.0)
    matchesMask = mask.ravel().tolist()

```

Với Shi-Tomasi Corner thì Nếu nó lớn hơn giá trị ngưỡng, nó được coi là một góc.



Code demo:

Tìm N góc mạnh nhất trong ảnh bằng phương pháp Shi-Tomasi (hoặc Harris Corner Detection, nếu bạn chỉ định nó). Như thường lệ, hình ảnh phải là một hình ảnh thang độ xám. Sau đó, bạn chỉ định số góc bạn muốn tìm. Sau đó, bạn chỉ định mức chất lượng, là giá trị từ 0-1, biểu thị chất lượng tối thiểu của góc bên dưới mà mọi người đều bị từ chối. Sau đó, chúng tôi cung cấp khoảng cách euclidean tối thiểu giữa các góc được phát hiện.

Với tất cả thông tin này, chức năng tìm thấy các góc trong hình ảnh. Tất cả các góc dưới mức chất lượng đều bị từ chối. Sau đó, nó sắp xếp các góc còn lại dựa trên chất lượng theo thứ tự giảm dần. Sau đó, hàm lấy góc mạnh nhất đầu tiên, vứt bỏ tất cả các góc gần đó trong phạm vi khoảng cách tối thiểu và trả về N góc mạnh nhất.

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
MIN_MATCH_COUNT = 10
img1 = cv.imread('img1.png', cv.IMREAD_GRAYSCALE)
img2 = cv.imread('img2.png', cv.IMREAD_GRAYSCALE)
sift = cv.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks = 50)
flann = cv.FlannBasedMatcher(index_params, search_params)
matches = flann.knnMatch(des1,des2,k=2)
# store all the good matches as per Lowe's ratio test.
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append(m)
if len(good)>MIN_MATCH_COUNT:
    src_pts = np.float32([ kp1[m.queryIdx].pt for m in good ]).reshape(-1,1,2)
    dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good ]).reshape(-1,1,2)
    M, mask = cv.findHomography(src_pts, dst_pts, cv.RANSAC,5.0)
    matchesMask = mask.ravel().tolist()
```

```

1 import numpy as np
2 import cv2 as cv
3 from matplotlib import pyplot as plt
4 img = cv.imread('img1.png')
5 gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
6 corners = cv.goodFeaturesToTrack(gray, 25, 0.01, 10)
7 corners = np.int0(corners)
8 for i in corners:
9     x,y = i.ravel()
10    cv.circle(img,(x,y),3,255,-1)
11 plt.imshow(img),plt.show()

```

for i in corners

\Programs\Python\Python310\python.exe "C:/Users/admin/Documents/

4) What is SIFT? Principle to find SIFT? Is it invariant to scaling, rotation, translation? A snippet code demo?

- SIFT(Biến đổi đặc trưng với tỷ lệ không bất biến): SIFT là thuật toán sử dụng để phát hiện và trích xuất đặc điểm trong hình ảnh và đặc biệt phổ biến trong các tác vụ như nhận dạng đối tượng, ghép ảnh và đăng ký hình ảnh. SIFT có khả năng trích xuất hình ảnh đặc biệt từ các điểm chính bất biến tỷ lệ, trích xuất các điểm chính và tính toán các mô tả của nó.
- Nguyên tắc tìm SIFT:
 - + Phát hiện cực trị không gian tỷ lệ:
 - + Bản địa hóa điểm chính: Sau khi tìm thấy các vị trí điểm chính tiềm năng, chúng phải được tinh chỉnh để có kết quả chính xác hơn. Họ đã sử dụng chuỗi mở rộng không gian tỷ lệ Taylor để có được vị trí cực trị chính xác hơn và nếu cường độ ở cực trị này nhỏ hơn giá trị ngưỡng (0,03 theo bài báo) thì nó sẽ bị từ chối. Ngưỡng này được gọi là **Ngưỡng tương phản** trong OpenCV

```

import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
MIN_MATCH_COUNT = 10
img1 = cv.imread('img1.png', cv.IMREAD_GRAYSCALE)
img2 = cv.imread('img2.png', cv.IMREAD_GRAYSCALE)
sift = cv.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks = 50)
flann = cv.FlannBasedMatcher(index_params, search_params)
matches = flann.knnMatch(des1,des2,k=2)
# store all the good matches as per Lowe's ratio test.
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append(m)
if len(good)>MIN_MATCH_COUNT:
    src_pts = np.float32([ kp1[m.queryIdx].pt for m in good ]).reshape(-1,1,2)
    dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good ]).reshape(-1,1,2)
    M, mask = cv.findHomography(src_pts, dst_pts, cv.RANSAC, 5.0)
    matchesMask = mask.ravel().tolist()

```

- + Bài tập định hướng: Một hướng được gán cho mỗi điểm chính để đạt được sự bất biến đối với việc xoay hình ảnh. Một vùng lân cận được lấy xung quanh vị trí điểm chính tùy thuộc vào tỷ lệ, đồng thời độ lớn và hướng của độ dốc được tính toán trong vùng đó
- + Mô tả điểm chính: Vùng lân cận 16x16 xung quanh điểm chính được lấy. Nó được chia thành 16 khối con có kích thước 4x4. Đối với mỗi khối con, biểu đồ định hướng 8 thùng được tạo. Vì vậy, có tổng cộng 128 giá trị bin. Nó được biểu diễn dưới dạng vector để tạo thành bộ mô tả điểm chính. Ngoài ra, một số biện pháp được thực hiện để đạt được độ bền chống lại sự thay đổi độ sáng, xoay, v.v.
- + So khớp điểm chính: Điểm chính giữa hai hình ảnh được so khớp bằng cách xác định các điểm lân cận gần nhất của chúng. Nhưng trong một số trường hợp, so khớp gần thứ hai có thể rất gần với so khớp đầu tiên. Nó có thể xảy ra do nhiều hoặc một số lý do khác. Trong trường hợp đó, tỷ lệ giữa khoảng cách gần nhất và khoảng cách gần thứ hai được lấy. Nếu lớn hơn 0,8 thì bị loại.
- SIFT có bất biến đối với việc chia tỷ lệ, xoay và dịch thông qua các bước gán hướng và phát hiện điểm chính của nó. Việc phát hiện cực trị không gian tỷ lệ và biểu diễn không gian tỷ lệ cho phép SIFT phát hiện các điểm chính ở các tỷ lệ khác nhau, làm cho nó bất biến trước những thay đổi về kích thước đối tượng (tỷ lệ). Bước gán hướng đảm bảo rằng các điểm chính được gán hướng nhất quán, làm cho SIFT bất biến đối với phép quay. Ngoài ra, tính bất biến dịch thuật đạt được bằng cách định vị các điểm chính trong vùng lân cận địa phương của chúng.
- Demo code:

```

import numpy as np
import cv2 as cv
img = cv.imread('img1.png')
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)

import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
MIN_MATCH_COUNT = 10
img1 = cv.imread('img1.png', cv.IMREAD_GRAYSCALE)
img2 = cv.imread('img2.png', cv.IMREAD_GRAYSCALE)
sift = cv.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks = 50)
flann = cv.FlannBasedMatcher(index_params, search_params)
matches = flann.knnMatch(des1,des2,k=2)
# store all the good matches as per Lowe's ratio test.
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append(m)
if len(good)>MIN_MATCH_COUNT:
    src_pts = np.float32([ kp1[m.queryIdx].pt for m in good ]).reshape(-1,1,2)
    dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good ]).reshape(-1,1,2)
    M, mask = cv.findHomography(src_pts, dst_pts, cv.RANSAC,5.0)
    matchesMask = mask.ravel().tolist()

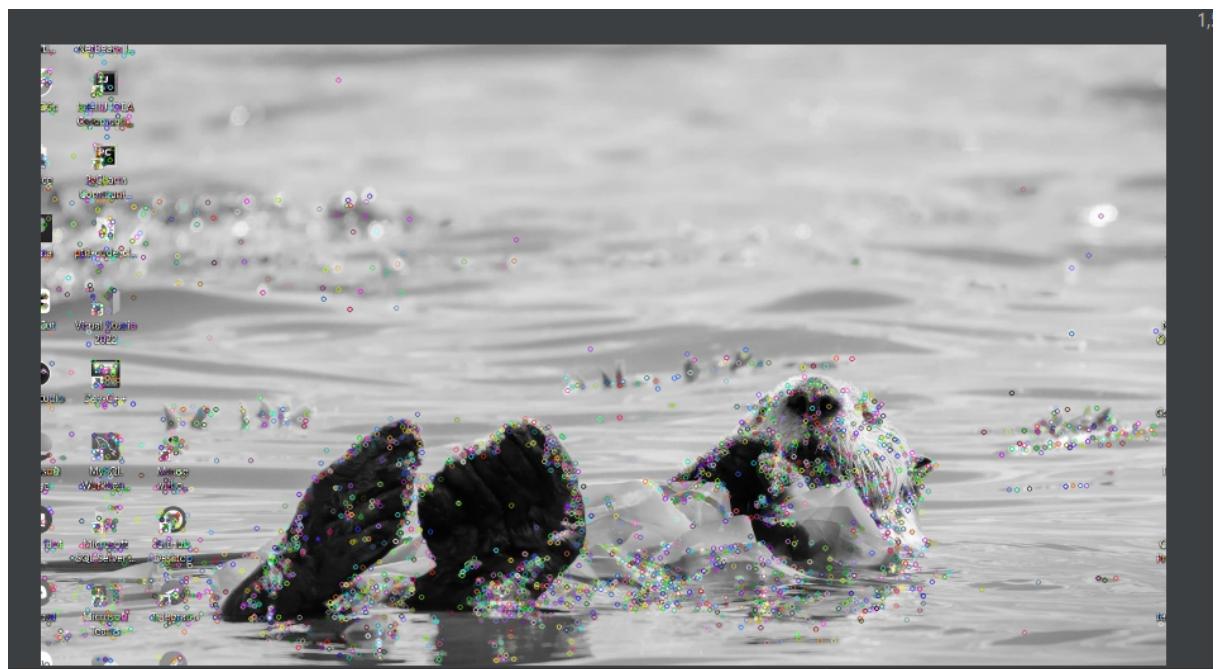
```

```

img=cv.drawKeypoints(gray,kp,img,
flags=cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
cv.imwrite('sift keypoints1.jpg',img)

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

```

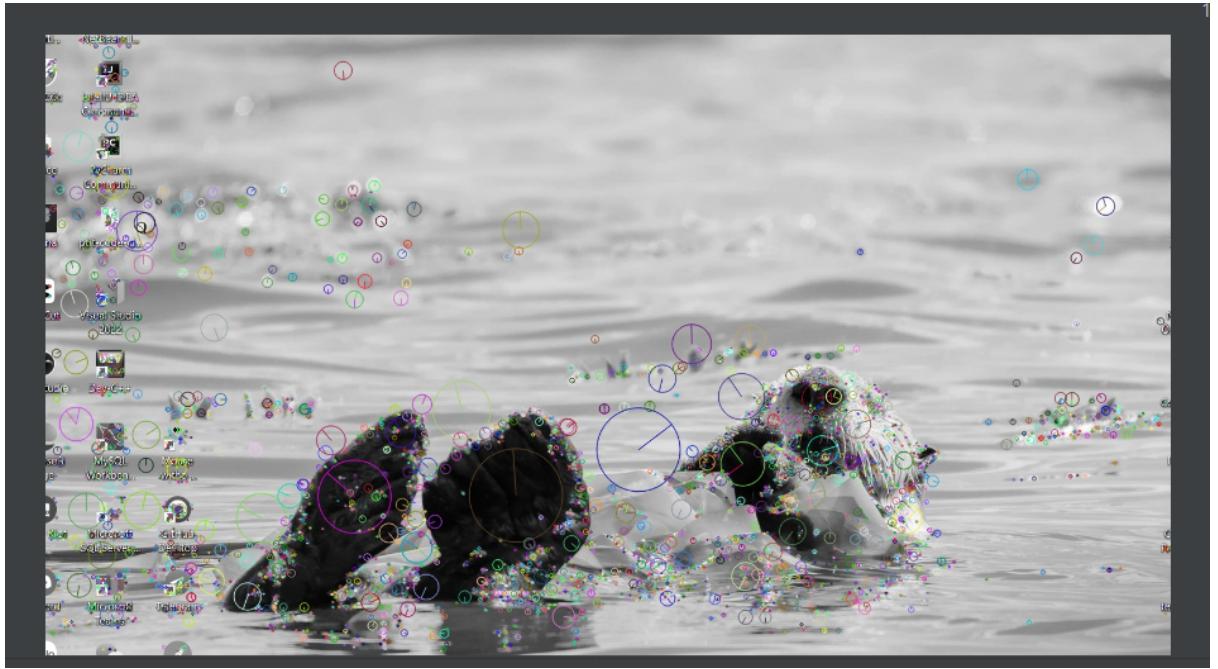


flag: DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS

```

import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
MIN_MATCH_COUNT = 10
img1 = cv.imread('img1.png', cv.IMREAD_GRAYSCALE)
img2 = cv.imread('img2.png', cv.IMREAD_GRAYSCALE)
sift = cv.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks = 50)
flann = cv.FlannBasedMatcher(index_params, search_params)
matches = flann.knnMatch(des1,des2,k=2)
# store all the good matches as per Lowe's ratio test.
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append(m)
if len(good)>MIN_MATCH_COUNT:
    src_pts = np.float32([ kp1[m.queryIdx].pt for m in good ]).reshape(-1,1,2)
    dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good ]).reshape(-1,1,2)
    M, mask = cv.findHomography(src_pts, dst_pts, cv.RANSAC,5.0)
    matchesMask = mask.ravel().tolist()

```



5) What is SURF? Principle to find SURF?

Is it invariant to scaling, rotation, translation? A snip code demo?

- SURF là thuật toán mà các đặc trưng được tăng cường mạnh mẽ. Đúng như tên gọi, đây là phiên bản tăng tốc của SIFT. SURF bổ sung rất nhiều tính năng để cải thiện tốc độ trong từng bước. Phân tích cho thấy nó nhanh hơn SIFT 3 lần trong khi hiệu suất tương đương với SIFT. SURF xử lý tốt các hình ảnh bị mờ và xoay, nhưng lại không giỏi xử lý sự thay đổi góc nhìn và thay đổi độ sáng.
- Tương tự như SIFT, SURF được thiết kế bất biến trước những thay đổi về tỷ lệ, góc quay và chuyển dịch. Nó được biết đến với hiệu quả tính toán, khiến nó phù hợp với các ứng dụng thời gian thực
- Demo code:

```

import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
MIN_MATCH_COUNT = 10
img1 = cv.imread('img1.png', cv.IMREAD_GRAYSCALE)
img2 = cv.imread('img2.png', cv.IMREAD_GRAYSCALE)
sift = cv.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks = 50)
flann = cv.FlannBasedMatcher(index_params, search_params)
matches = flann.knnMatch(des1,des2,k=2)
# store all the good matches as per Lowe's ratio test.
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append(m)
if len(good)>MIN_MATCH_COUNT:
    src_pts = np.float32([ kp1[m.queryIdx].pt for m in good ]).reshape(-1,1,2)
    dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good ]).reshape(-1,1,2)
    M, mask = cv.findHomography(src_pts, dst_pts, cv.RANSAC,5.0)
    matchesMask = mask.ravel().tolist()

```

6) Brief introduction to FAST Algorithm for Corner Detection? a snip code demo

- Thuật toán phát hiện đặc trưng FAST gồm các bước sau:
 - + Chọn một pixel p trong hình ảnh có được xác định là điểm quan tâm hay không. Hãy để cường độ của nó là I_p .
 - + Chọn giá trị ngưỡng thích hợp t
 - + Hãy xem xét một vòng tròn 16 pixel xung quanh pixel đang được thử nghiệm.
 - + Bây giờ pixel P là một góc nếu tồn tại một tập hợp n các pixel liền kề trong vòng tròn (16 pixel) đều sáng hơn $I_p + t$, hoặc tối hơn tất cả $I_p - t$. (Hiển thị dưới dạng đường gạch ngang màu trắng trong hình trên), n được chọn là 12.
 - + Một thử nghiệm tốc độ cao đã được đề xuất để loại trừ một số lượng lớn các điểm không phải biên. Thử nghiệm này chỉ kiểm tra bốn pixel ở 1, 9, 5 và 13 (1 và 9 đầu tiên được kiểm tra xem chúng quá sáng hơn hay tối hơn. Nếu vậy thì kiểm tra 5 và 13). Nếu p là góc, ít nhất ba trong số đó phải sáng hơn $I_p + t$ hoặc tối hơn $I_p - t$. Nếu cả hai điều này đều không xảy ra thì p không thể là biên. Sau đó, tiêu chí kiểm tra phân đoạn đầy đủ có thể được áp dụng cho các ứng viên đã đạt bằng cách kiểm tra tất cả các pixel trong vòng tròn. Bản thân máy dò này thể hiện hiệu suất cao, nhưng có một số điểm yếu:
 - . Nó không từ chối nhiều ứng cử viên cho $n < 12$.
 - . Việc lựa chọn các pixel không phải là tối ưu vì hiệu quả của nó phụ thuộc vào thứ tự của các câu hỏi và sự phân bổ các góc xuất hiện.
 - . Kết quả kiểm tra tốc độ cao bị vứt đi.
 - . Nhiều tính năng được phát hiện liền kề nhau.

- Nó nhanh hơn nhiều lần so với các máy dò góc hiện có khác.
- Nhưng nó không bền với mức độ ồn cao. Nó phụ thuộc vào một ngưỡng.
- Demo code:

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

import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
MIN_MATCH_COUNT = 10
img1 = cv.imread('img1.png', cv.IMREAD_GRAYSCALE)
img2 = cv.imread('img2.png', cv.IMREAD_GRAYSCALE)
sift = cv.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks = 50)
flann = cv.FlannBasedMatcher(index_params, search_params)
matches = flann.knnMatch(des1,des2,k=2)
# store all the good matches as per Lowe's ratio test.
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append(m)
if len(good)>MIN_MATCH_COUNT:
    src_pts = np.float32([ kp1[m.queryIdx].pt for m in good ]).reshape(-1,1,2)
    dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good ]).reshape(-1,1,2)
    M, mask = cv.findHomography(src_pts, dst_pts, cv.RANSAC, 5.0)
    matchesMask = mask.ravel().tolist()
```

```

img      = cv.imread('anh1.png',           cv.IMREAD_GRAYSCALE)    #
`<opencv_root>/samples/data/blox.jpg`
# Initiate FAST object with default values
fast = cv.FastFeatureDetector_create()
# find and draw the keypoints
kp = fast.detect(img, None)
img2 = cv.drawKeypoints(img, kp, None, color=(255,0,0))
# Print all default params
print("Threshold: {}".format(fast.getThreshold()))
print("nonmaxSuppression:{}".format(fast.getNonmaxSuppression()))
print("neighborhood: {}".format(fast.getType()))
print("Total Keypoints with nonmaxSuppression: {}".format(len(kp)))
cv.imwrite('fast_true.png', img2)
# Disable nonmaxSuppression
fast.setNonmaxSuppression(0)
kp = fast.detect(img, None)
print("Total Keypoints without nonmaxSuppression: {}".format(len(kp)))
img3 = cv.drawKeypoints(img, kp, None, color=(255,0,0))
cv.imwrite('fast_false.png', img3)

```

Output:

```

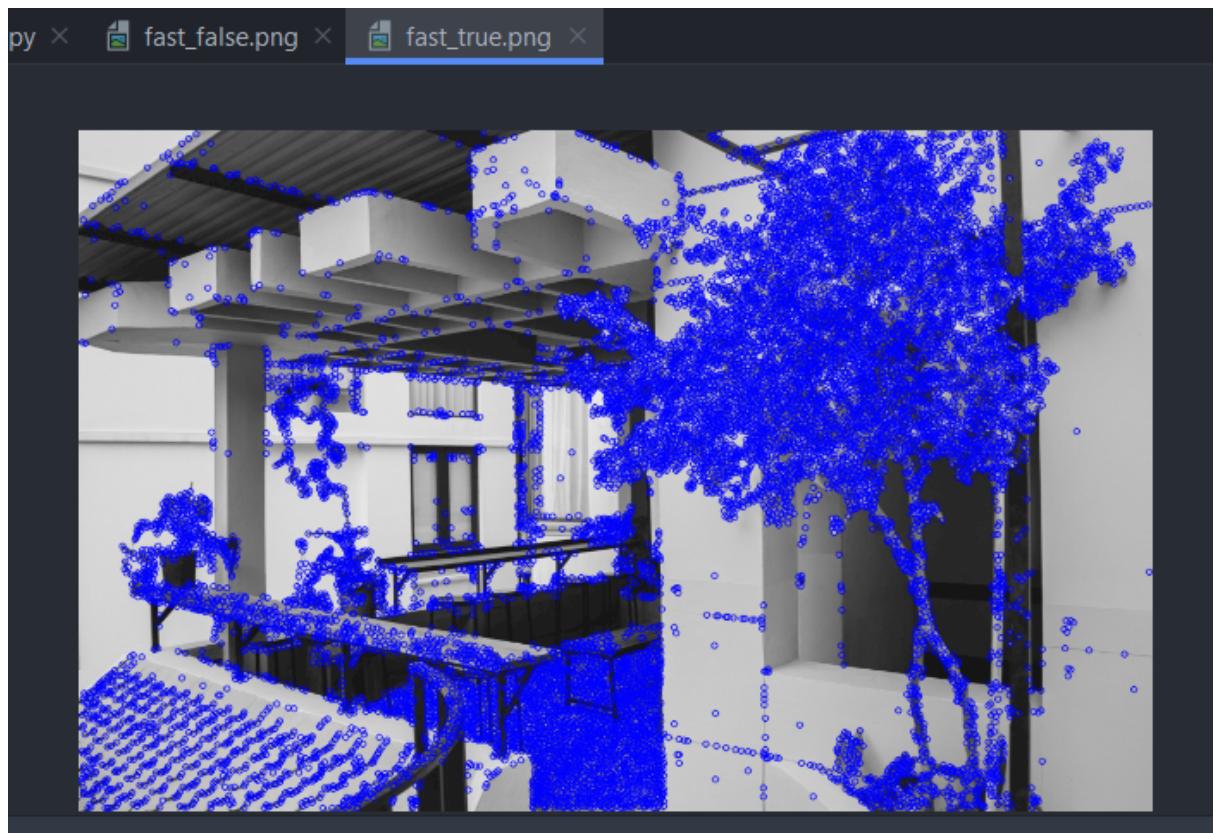
Threshold: 10
nonmaxSuppression:True
neighborhood: 2
Total Keypoints with nonmaxSuppression: 14633
Total Keypoints without nonmaxSuppression: 71730

```

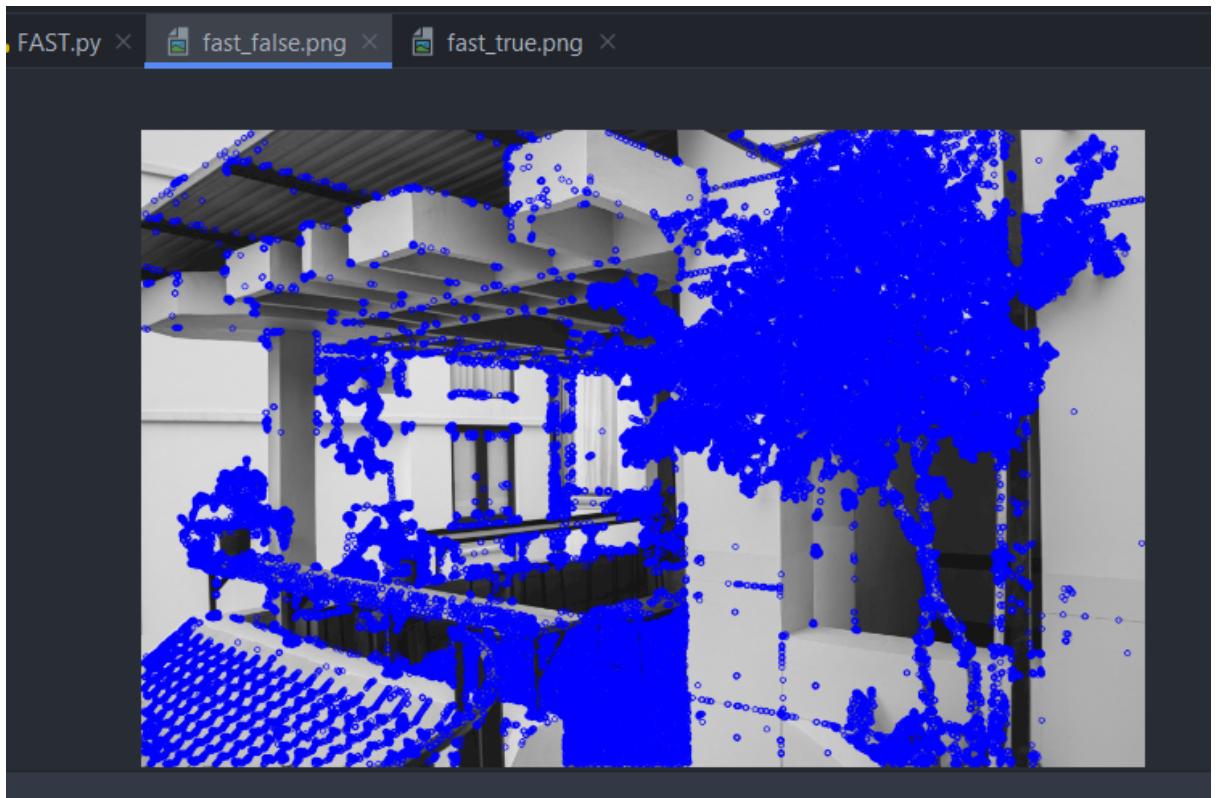
```

import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
MIN_MATCH_COUNT = 10
img1 = cv.imread('img1.png', cv.IMREAD_GRAYSCALE)
img2 = cv.imread('img2.png', cv.IMREAD_GRAYSCALE)
sift = cv.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks = 50)
flann = cv.FlannBasedMatcher(index_params, search_params)
matches = flann.knnMatch(des1,des2,k=2)
# store all the good matches as per Lowe's ratio test.
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append(m)
if len(good)>MIN_MATCH_COUNT:
    src_pts = np.float32([ kp1[m.queryIdx].pt for m in good ]).reshape(-1,1,2)
    dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good ]).reshape(-1,1,2)
    M, mask = cv.findHomography(src_pts, dst_pts, cv.RANSAC, 5.0)
    matchesMask = mask.ravel().tolist()

```



```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
MIN_MATCH_COUNT = 10
img1 = cv.imread('img1.png', cv.IMREAD_GRAYSCALE)
img2 = cv.imread('img2.png', cv.IMREAD_GRAYSCALE)
sift = cv.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks = 50)
flann = cv.FlannBasedMatcher(index_params, search_params)
matches = flann.knnMatch(des1,des2,k=2)
# store all the good matches as per Lowe's ratio test.
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append(m)
if len(good)>MIN_MATCH_COUNT:
    src_pts = np.float32([ kp1[m.queryIdx].pt for m in good ]).reshape(-1,1,2)
    dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good ]).reshape(-1,1,2)
    M, mask = cv.findHomography(src_pts, dst_pts, cv.RANSAC,5.0)
    matchesMask = mask.ravel().tolist()
```



7) What is BRIEF? a snip code demo?

- BRIEF cung cấp một lối tắt để tìm trực tiếp các chuỗi nhị phân mà không cần tìm bộ mô tả. Nó lấy bản vá hình ảnh được làm mịn và chọn một tập hợp $N_d(x,y)$ cặp vị trí theo cách duy nhất (được giải thích trên giấy). Sau đó, một số so sánh cường độ pixel được thực hiện trên các cặp vị trí này. Ví dụ: đặt cặp vị trí đầu tiên là p và q/ Nếu $I(p) < I(q)$ thì kết quả là 1, nếu không kết quả là 0. Điều này được áp dụng cho tất cả các N_d cặp vị trí để có được một N_d -chuỗi bit chiều.

```

import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
MIN_MATCH_COUNT = 10
img1 = cv.imread('img1.png', cv.IMREAD_GRAYSCALE)
img2 = cv.imread('img2.png', cv.IMREAD_GRAYSCALE)
sift = cv.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks = 50)
flann = cv.FlannBasedMatcher(index_params, search_params)
matches = flann.knnMatch(des1,des2,k=2)
# store all the good matches as per Lowe's ratio test.
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append(m)
if len(good)>MIN_MATCH_COUNT:
    src_pts = np.float32([ kp1[m.queryIdx].pt for m in good ]).reshape(-1,1,2)
    dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good ]).reshape(-1,1,2)
    M, mask = cv.findHomography(src_pts, dst_pts, cv.RANSAC, 5.0)
    matchesMask = mask.ravel().tolist()

```

- BRIEF là một phương pháp tính toán và so khớp mô tả đặc trưng phương pháp nhanh hơn. Nó cũng cung cấp tỷ lệ nhận dạng cao trừ khi có chuyển động quay trong mặt phẳng lớn.
- BRIEF là một bộ mô tả đặc trưng, nó không cung cấp bất kỳ phương pháp nào để tìm các đặc trưng.
- Demo code:

```
import cv2 as cv
img = cv.imread('img1.png', cv.IMREAD_GRAYSCALE)
star = cv.xfeatures2d.StarDetector_create()
brief = cv.xfeatures2d.BriefDescriptorExtractor_create()
kp = star.detect(img, None)
kp, des = brief.compute(img, kp)
print('Descriptor Size: ', brief.descriptorSize())
print('Descriptor Shape', des.shape)
```

Output:

```
C:\Users\admin\AppData\Local\Programs\Python\Py
Descriptor Size: 32
Descriptor Shape (835, 32)
```

8) What is ORB? a snip code demo?

- ORB về cơ bản là sự kết hợp giữa bộ phát hiện điểm khóa FAST và bộ mô tả BRIEF với nhiều sửa đổi để nâng cao hiệu suất. Đầu tiên, nó sử dụng FAST để tìm các điểm chính, sau đó áp dụng thước đo góc Harris để tìm N điểm cao nhất trong số đó. Nó cũng sử dụng kim tự tháp để tạo ra các tính năng đa tỷ lệ. Nhưng có một vấn đề là FAST không tính toán được hướng.
- Nó tính toán trọng tâm cường độ của miếng vá có góc nằm ở giữa. Hướng của vectơ từ điểm góc này đến tâm sẽ cho biết hướng. Để cải thiện tính bất biến của phép quay, các

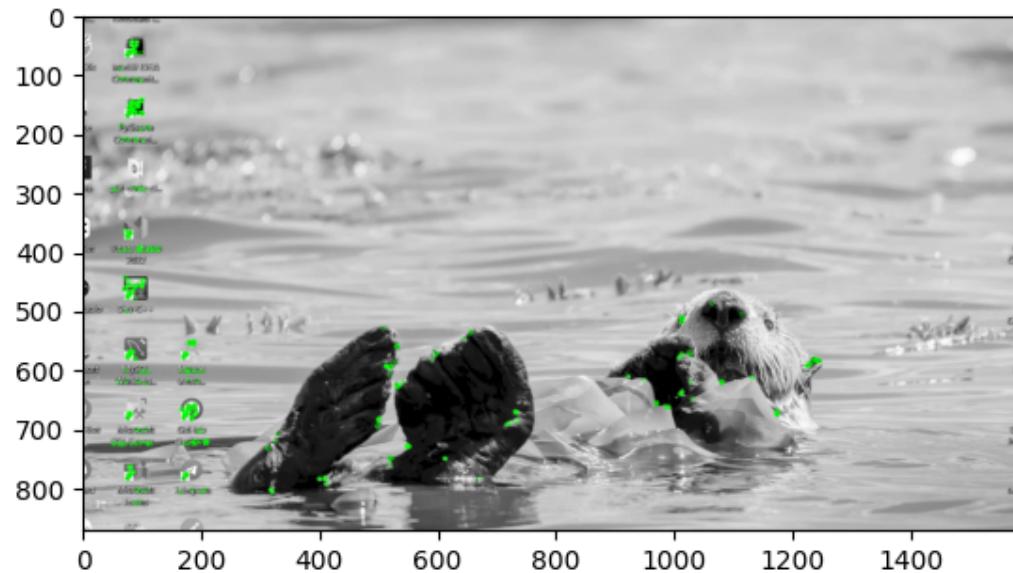
```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
MIN_MATCH_COUNT = 10
img1 = cv.imread('img1.png', cv.IMREAD_GRAYSCALE)
img2 = cv.imread('img2.png', cv.IMREAD_GRAYSCALE)
sift = cv.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks = 50)
flann = cv.FlannBasedMatcher(index_params, search_params)
matches = flann.knnMatch(des1,des2,k=2)
# store all the good matches as per Lowe's ratio test.
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append(m)
if len(good)>MIN_MATCH_COUNT:
    src_pts = np.float32([ kp1[m.queryIdx].pt for m in good ]).reshape(-1,1,2)
    dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good ]).reshape(-1,1,2)
    M, mask = cv.findHomography(src_pts, dst_pts, cv.RANSAC, 5.0)
    matchesMask = mask.ravel().tolist()
```

khoanh khắc được tính bằng x và y phải nằm trong vùng bán kính hình tròn r . Ở đây, r là kích thước của bản vá.

- ORB sử dụng các bộ mô tả BRIEF. Nhưng chúng ta đã thấy rằng BRIEF hoạt động kém khi xoay. Vì vậy, điều ORB làm là “điều khiển” BRIEF theo định hướng của các điểm chính. Đối với bất kỳ đặc trưng nào của n thử nghiệm nhị phân tại vị trí (x,y) sẽ định nghĩa 1 ma trận S $2 \times n$, cái sẽ chứa các tọa độ này. Sau đó, sử dụng hướng của bản vá, θ , ma trận xoay của nó được tìm thấy và quay S để có phiên bản được điều khiển (xoay) $S\theta$
- ORB rời rạc hóa góc theo giá số của $2\pi/30$ (12 độ) và xây dựng bảng tra cứu các mẫu BRIEF được tính toán trước. Miễn là định hướng điểm chính θ nhất quán giữa các quan điểm, tập hợp các điểm chính xác $S\theta$ sẽ được sử dụng để tính toán mô tả của nó.
- ORB nhanh hơn nhiều so với SURF và SIFT và bộ mô tả ORB hoạt động tốt hơn SURF. ORB là một lựa chọn tốt trong các thiết bị tiêu thụ điện năng thấp để ghép ảnh toàn cảnh, v.v.
- Demo code:

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('img1.png', cv.IMREAD_GRAYSCALE)
orb = cv.ORB_create()
kp = orb.detect(img, None)
kp, des = orb.compute(img, kp)
img2 = cv.drawKeypoints(img, kp, None, color=(0, 255, 0), flags=0)
plt.imshow(img2), plt.show()
```

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
MIN_MATCH_COUNT = 10
img1 = cv.imread('img1.png', cv.IMREAD_GRAYSCALE)
img2 = cv.imread('img2.png', cv.IMREAD_GRAYSCALE)
sift = cv.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks = 50)
flann = cv.FlannBasedMatcher(index_params, search_params)
matches = flann.knnMatch(des1,des2,k=2)
# store all the good matches as per Lowe's ratio test.
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append(m)
if len(good)>MIN_MATCH_COUNT:
    src_pts = np.float32([ kp1[m.queryIdx].pt for m in good ]).reshape(-1,1,2)
    dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good ]).reshape(-1,1,2)
    M, mask = cv.findHomography(src_pts, dst_pts, cv.RANSAC, 5.0)
    matchesMask = mask.ravel().tolist()
```



9) Idea(s) behind Feature Matching? a snip code demo?

```

import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
MIN_MATCH_COUNT = 10
img1 = cv.imread('img1.png', cv.IMREAD_GRAYSCALE)
img2 = cv.imread('img2.png', cv.IMREAD_GRAYSCALE)
sift = cv.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks = 50)
flann = cv.FlannBasedMatcher(index_params, search_params)
matches = flann.knnMatch(des1,des2,k=2)
# store all the good matches as per Lowe's ratio test.
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append(m)
if len(good)>MIN_MATCH_COUNT:
    src_pts = np.float32([ kp1[m.queryIdx].pt for m in good ]).reshape(-1,1,2)
    dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good ]).reshape(-1,1,2)
    M, mask = cv.findHomography(src_pts, dst_pts, cv.RANSAC,5.0)
    matchesMask = mask.ravel().tolist()

```

a) Brute-Force Basic:

Brute-Force matcher rất đơn giản. Nó lấy mô tả của một tính năng trong bộ đầu tiên và được khớp với tất cả các tính năng khác trong bộ thứ hai bằng cách sử dụng một số tính toán khoảng cách. Và cái gần nhất được trả lại.

Đối với BF matcher, đầu tiên chúng ta phải tạo đối tượng BFMatcher bằng `cv.BFMatcher()`. Phải mất hai tham số tùy chọn. Đầu tiên là `normType`. Nó chỉ định phép đo khoảng cách sẽ được sử dụng. Theo mặc định, nó là `cv.NORM_L2`. Nó tốt cho SIFT, SURF, v.v. (`cv.NORM_L1` cũng ở đó). Đối với các mô tả dựa trên chuỗi nhị phân như ORB, BRIEF, BRISK, v.v., `cv.NORM_HAMMING` nên được sử dụng, sử dụng khoảng cách Hamming làm phép đo. Nếu ORB đang sử dụng `WTA_K == 3` hoặc `4`, `cv.NORM_HAMMING2` nên được sử dụng.

Tham số thứ hai là biến boolean, `crossCheck` là `false` theo mặc định. Nếu đúng, Matcher chỉ trả về những kết quả phù hợp với giá trị (i,j) sao cho mô tả thứ i trong tập A có mô tả thứ j trong tập B là khớp tốt nhất và ngược lại. Đó là, hai tính năng trong cả hai bộ phải khớp với nhau. Nó cung cấp kết quả nhất quán, và là một thay thế tốt cho thử nghiệm tỷ lệ được đề xuất bởi D.Lowe trong giấy SIFT.

Khi nó được tạo, hai phương thức quan trọng là `BFMatcher.match()` và `BFMatcher.knnMatch()`. Đầu tiên trả về trận đấu tốt nhất. Phương thức thứ hai trả về k kết quả phù hợp nhất trong đó k được chỉ định bởi người dùng. Nó có thể hữu ích khi chúng ta cần làm thêm công việc về điều đó.

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
MIN_MATCH_COUNT = 10
img1 = cv.imread('img1.png', cv.IMREAD_GRAYSCALE)
img2 = cv.imread('img2.png', cv.IMREAD_GRAYSCALE)
sift = cv.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks = 50)
flann = cv.FlannBasedMatcher(index_params, search_params)
matches = flann.knnMatch(des1,des2,k=2)
# store all the good matches as per Lowe's ratio test.
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append(m)
if len(good)>MIN_MATCH_COUNT:
    src_pts = np.float32([ kp1[m.queryIdx].pt for m in good ]).reshape(-1,1,2)
    dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good ]).reshape(-1,1,2)
    M, mask = cv.findHomography(src_pts, dst_pts, cv.RANSAC,5.0)
    matchesMask = mask.ravel().tolist()
```

Giống như chúng ta đã sử dụng cv.drawKeypoints() để vẽ keypoints, cv.drawMatches() giúp chúng ta vẽ các trận đấu. Nó xếp chồng hai hình ảnh theo chiều ngang và vẽ các đường từ hình ảnh đầu tiên đến hình ảnh thứ hai hiển thị các kết quả phù hợp nhất. Ngoài ra còn có cv.drawMatchesKnn rút thăm tất cả k trận đấu hay nhất. Nếu k = 2, nó sẽ vẽ hai đường khớp cho mỗi điểm chính. Vì vậy, chúng ta phải vượt qua một mặt nạ nếu chúng ta muốn vẽ nó một cách có chọn lọc.

b) Brute-Force Matching với ORB Descriptors:

Ở đây, chúng ta sẽ thấy một ví dụ đơn giản về cách kết hợp các tính năng giữa hai hình ảnh. Trong trường hợp này, tôi có một queryImage và một trainImage. Chúng ta sẽ cố gắng tìm queryImage trong trainImage bằng cách sử dụng feature matching.

Kết quả của dòng = bf.match(des1,des2) là danh sách các đối tượng DMatch. Đối tượng DMatch này có các thuộc tính sau:

- DMatch.distance - Khoảng cách giữa các mô tả. Càng thấp, càng tốt.
- DMatch.trainIdx - Chỉ mục mô tả trong mô tả đối tượng
- DMatch.queryIdx - Chỉ mục mô tả trong mô tả truy vấn
- DMatch.imgIdx - Chỉ mục của hình ảnh đối tượng.

Code Demo:

```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
img1 = cv.imread('img1.png', cv.IMREAD_GRAYSCALE)
img2 = cv.imread('img2.png', cv.IMREAD_GRAYSCALE) # trainImage
orb = cv.ORB_create()
```

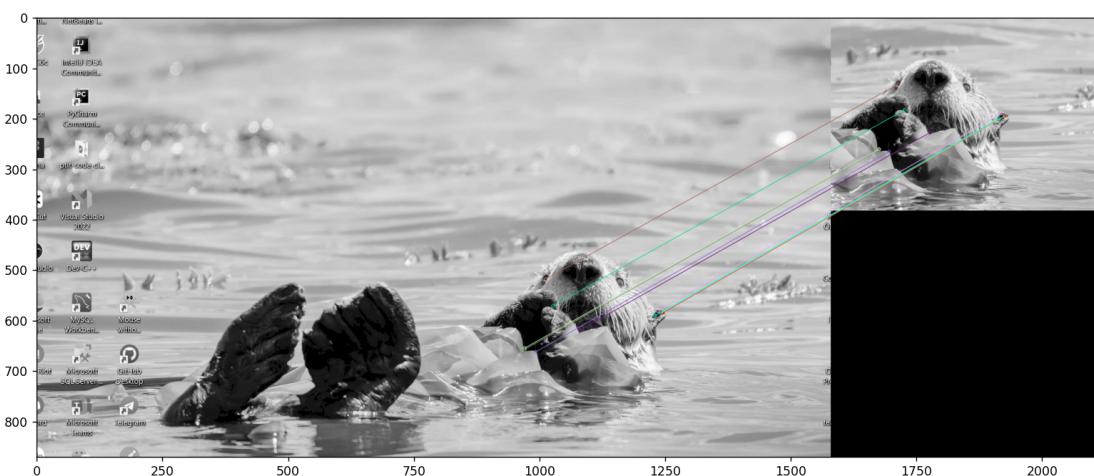
```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
MIN_MATCH_COUNT = 10
img1 = cv.imread('img1.png', cv.IMREAD_GRAYSCALE)
img2 = cv.imread('img2.png', cv.IMREAD_GRAYSCALE)
sift = cv.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks = 50)
flann = cv.FlannBasedMatcher(index_params, search_params)
matches = flann.knnMatch(des1,des2,k=2)
# store all the good matches as per Lowe's ratio test.
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append(m)
if len(good)>MIN_MATCH_COUNT:
    src_pts = np.float32([ kp1[m.queryIdx].pt for m in good ]).reshape(-1,1,2)
    dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good ]).reshape(-1,1,2)
    M, mask = cv.findHomography(src_pts, dst_pts, cv.RANSAC,5.0)
    matchesMask = mask.ravel().tolist()
```

```

kp1, des1 = orb.detectAndCompute(img1, None)
kp2, des2 = orb.detectAndCompute(img2, None)

bf = cv.BFMatcher(cv.NORM_HAMMING, crossCheck=True)
matches = bf.match(des1,des2)
matches = sorted(matches, key = lambda x:x.distance)
img3 = cv.drawMatches(img1,kp1,img2,kp2,matches[:10],None,flags=cv.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
plt.imshow(img3)
plt.show()

```



c) Brute-Force Matching with SIFT Descriptors and Ratio Test

```

import numpy as np
import cv2 as cv

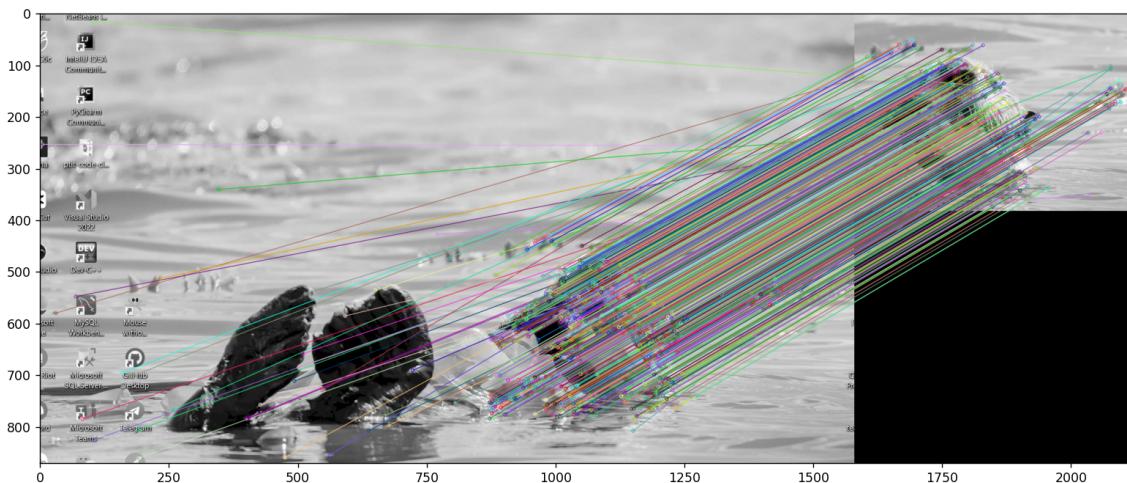
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
MIN_MATCH_COUNT = 10
img1 = cv.imread('img1.png', cv.IMREAD_GRAYSCALE)
img2 = cv.imread('img2.png', cv.IMREAD_GRAYSCALE)
sift = cv.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks = 50)
flann = cv.FlannBasedMatcher(index_params, search_params)
matches = flann.knnMatch(des1,des2,k=2)
# store all the good matches as per Lowe's ratio test.
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append(m)
if len(good)>MIN_MATCH_COUNT:
    src_pts = np.float32([ kp1[m.queryIdx].pt for m in good ]).reshape(-1,1,2)
    dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good ]).reshape(-1,1,2)
    M, mask = cv.findHomography(src_pts, dst_pts, cv.RANSAC,5.0)
    matchesMask = mask.ravel().tolist()

```

```

import matplotlib.pyplot as plt
img1 = cv.imread('img1.png',cv.IMREAD_GRAYSCALE) # queryImage
img2 = cv.imread('img2.png',cv.IMREAD_GRAYSCALE) # trainImage
sift = cv.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1,None)
kp2, des2 = sift.detectAndCompute(img2,None)
bf = cv.BFMatcher()
matches = bf.knnMatch(des1,des2,k=2)
good = []
for m,n in matches:
    if m.distance < 0.75*n.distance:
        good.append([m])
img3 = cv.drawMatchesKnn(img1,kp1,img2,kp2,good,None,flags=cv.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
plt.imshow(img3),plt.show()

```



```

import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
MIN_MATCH_COUNT = 10
img1 = cv.imread('img1.png', cv.IMREAD_GRAYSCALE)
img2 = cv.imread('img2.png', cv.IMREAD_GRAYSCALE)
sift = cv.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks = 50)
flann = cv.FlannBasedMatcher(index_params, search_params)
matches = flann.knnMatch(des1,des2,k=2)
# store all the good matches as per Lowe's ratio test.
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append(m)
if len(good)>MIN_MATCH_COUNT:
    src_pts = np.float32([ kp1[m.queryIdx].pt for m in good ]).reshape(-1,1,2)
    dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good ]).reshape(-1,1,2)
    M, mask = cv.findHomography(src_pts, dst_pts, cv.RANSAC,5.0)
    matchesMask = mask.ravel().tolist()

```

d) FLANN based Matcher

FLANN là viết tắt của Thư viện nhanh cho những người hàng xóm gần nhất gần đúng. Nó chứa một bộ sưu tập các thuật toán được tối ưu hóa để tìm kiếm hàng xóm gần nhất nhanh chóng trong các bộ dữ liệu lớn và cho các tính năng chiều cao. Nó hoạt động nhanh hơn BFMatcher cho các bộ dữ liệu lớn. Chúng ta sẽ thấy ví dụ thứ hai với trình kết hợp dựa trên FLANN.

Đối với trình kết hợp dựa trên FLANN, chúng ta cần chuyển hai từ điển chỉ định thuật toán sẽ được sử dụng, các tham số liên quan của nó, v.v. Đầu tiên là IndexParams. Đối với các thuật toán khác nhau, thông tin được truyền được giải thích trong tài liệu FLANN. Tóm lại, đối với các thuật toán như SIFT, SURF, v.v., bạn có thể vượt qua những điều sau:

Trong khi sử dụng ORB, bạn có thể vượt qua những điều sau đây. Các giá trị nhận xét được đề xuất theo tài liệu, nhưng nó không cung cấp kết quả bắt buộc trong một số trường hợp. Các giá trị khác hoạt động tốt.:

```
FLANN_INDEX_LSH = 6
```

Tùy diễn thứ hai là SearchParams. Nó chỉ định số lần các cây trong chỉ mục nên được đi qua đệ quy. Giá trị cao hơn cho độ chính xác tốt hơn, nhưng cũng mất nhiều thời gian hơn. Nếu bạn muốn thay đổi giá trị, hãy truyền search_params = dict(checks=100).

Code Demo:

```
import numpy as np
```

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
MIN_MATCH_COUNT = 10
img1 = cv.imread('img1.png', cv.IMREAD_GRAYSCALE)
img2 = cv.imread('img2.png', cv.IMREAD_GRAYSCALE)
sift = cv.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks = 50)
flann = cv.FlannBasedMatcher(index_params, search_params)
matches = flann.knnMatch(des1,des2,k=2)
# store all the good matches as per Lowe's ratio test.
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append(m)
if len(good)>MIN_MATCH_COUNT:
    src_pts = np.float32([ kp1[m.queryIdx].pt for m in good ])
    src_pts.reshape(-1,1,2)
    dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good ])
    dst_pts.reshape(-1,1,2)
    M, mask = cv.findHomography(src_pts, dst_pts, cv.RANSAC, 5.0)
    matchesMask = mask.ravel().tolist()
```

```

import cv2 as cv

import matplotlib.pyplot as plt

img1 = cv.imread('img1.png',cv.IMREAD_GRAYSCALE)

img2 = cv.imread('img2.png',cv.IMREAD_GRAYSCALE)

sift = cv.SIFT_create()

kp1, des1 = sift.detectAndCompute(img1,None)

kp2, des2 = sift.detectAndCompute(img2,None)

FLANN_INDEX_KDTREE = 1

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

search_params = dict(checks=50)

flann = cv.FlannBasedMatcher(index_params,search_params)

matches = flann.knnMatch(des1,des2,k=2)

matchesMask = [[0,0] for i in range(len(matches))]

for i,(m,n) in enumerate(matches):

    if m.distance < 0.7*n.distance:

        matchesMask[i]=[1,0]

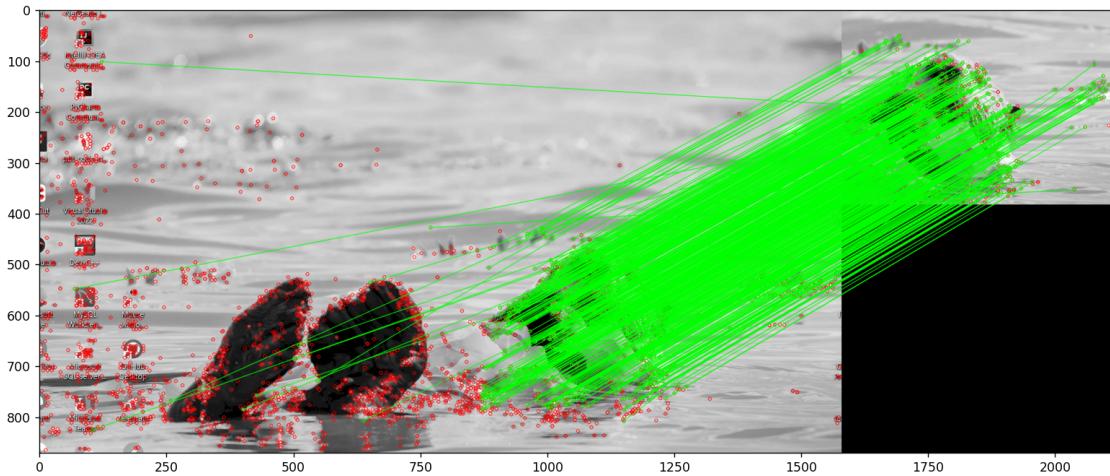
draw_params = dict(matchColor = (0,255,0), singlePointColor = (255,0,0),
                   matchesMask = matchesMask, flags = cv.DrawMatchesFlags_DEFAULT)

img3 = cv.drawMatchesKnn(img1,kp1,img2,kp2,matches,None,**draw_params)

import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
MIN_MATCH_COUNT = 10
img1 = cv.imread('img1.png', cv.IMREAD_GRAYSCALE)
img2 = cv.imread('img2.png', cv.IMREAD_GRAYSCALE)
sift = cv.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks = 50)
flann = cv.FlannBasedMatcher(index_params, search_params)
matches = flann.knnMatch(des1,des2,k=2)
# store all the good matches as per Lowe's ratio test.
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append(m)
if len(good)>MIN_MATCH_COUNT:
    src_pts = np.float32([ kp1[m.queryIdx].pt for m in good ]).reshape(-1,1,2)
    dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good ]).reshape(-1,1,2)
    M, mask = cv.findHomography(src_pts, dst_pts, cv.RANSAC,5.0)
    matchesMask = mask.ravel().tolist()

```

```
plt.imshow(img3), plt.show()
```



10) Application: recover the example Feature Matching + Homography to find Objects

- Demo code:

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
MIN_MATCH_COUNT = 10
img1 = cv.imread('img1.png', cv.IMREAD_GRAYSCALE)
img2 = cv.imread('img2.png', cv.IMREAD_GRAYSCALE)
sift = cv.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks = 50)
flann = cv.FlannBasedMatcher(index_params, search_params)
matches = flann.knnMatch(des1,des2,k=2)
# store all the good matches as per Lowe's ratio test.
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append(m)
if len(good)>MIN_MATCH_COUNT:
    src_pts = np.float32([ kp1[m.queryIdx].pt for m in good ]).reshape(-1,1,2)
    dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good ]).reshape(-1,1,2)
    M, mask = cv.findHomography(src_pts, dst_pts, cv.RANSAC, 5.0)
    matchesMask = mask.ravel().tolist()
```