Brute-Force Matcher:

کلاس **BFMatcher** در کتابخانه‌ی OpenCV ، اختصار عبارت **Brute-Force Matcher** است و وظیفه‌ی آن مقایسه‌ی مستقیم توصیف‌گرهای (Descriptors) استخراج‌شده از تصاویر مختلف است. این مقایسه به‌صورت جامع انجام می‌شود؛ بدین معنا که هر توصیف‌گر از تصویر اول با تمامی توصیف‌گرهای تصویر دوم مقایسه شده و نزدیک‌ترین جفت بر اساس معیار فاصله انتخاب می‌گردد. از این رو این روش ساده، شفاف و دقیق است، اما از لحاظ محاسباتی ممکن است برای داده‌های بسیار بزرگ زمان‌بر باشد.

ورودی نخست `normType` :

پارامتر **`normType`** نوع معیار فاصله‌ای را مشخص می‌سازد که برای مقایسه‌ی توصیف‌گرها به‌کار گرفته می‌شود. انتخاب این معیار به ماهیت توصیف‌گر بستگی دارد:

- در صورتی که توصیف‌گرها از نوع باینری باشند مانند ORB ، BRIEF یا BRISK ، معیار مناسب **Hamming Distance** است که اختلاف بین بیت‌های دو بردار را محاسبه می‌کند.
- در مواردی که توصیف‌گرها شامل مقادیر حقیقی یا اعشاری باشند مانند SIFT و SURF ، معیارهایی همچون **L1 norm (Manhattan Distance)** یا **L2 norm (Euclidean Distance)** مورد استفاده قرار می‌گیرند. انتخاب صحیح این پارامتر تأثیر مستقیمی بر کیفیت تطبیق دارد.
- **L1 (Manhattan Distance)** فاصله‌ای است که با جمع قدرمطلق اختلاف مختصات محاسبه می‌شود. می‌توان آن را به پیمودن مسیر در خیابان‌های شطرنجی تشبیه کرد، جایی که فقط حرکت افقی و عمودی مجاز است.
- **L2 (Euclidean Distance)** فاصله‌ی اقلیدسی همان کوتاه‌ترین خط مستقیم بین دو نقطه در فضا است. این معیار متداول‌ترین روش برای سنجش شباهت و فاصله‌ی هندسی بین بردارها به شمار می‌رود.

ورودی دوم `crossCheck` :

پارامتر **`crossCheck`** تعیین می‌کند که فرآیند تطبیق به صورت یک‌طرفه یا دوطرفه انجام گیرد.

- در حالت پیش‌فرض که مقدار آن **False** است، تطبیق یک‌طرفه انجام می‌شود؛ بدین معنا که اگر توصیف‌گر A در تصویر اول نزدیک‌ترین همتای B در تصویر دوم باشد، این جفت به‌عنوان Match پذیرفته می‌شود، حتی اگر عکس آن صادق نباشد.

- در مقابل، اگر مقدار **True** انتخاب گردد، تطبیق تنها زمانی معتبر خواهد بود که رابطه‌ی نزدیکی متقابل برقرار باشد؛ یعنی A نزدیک‌ترین به B و B نزدیک‌ترین به A باشد. این حالت موجب افزایش اطمینان و کاهش خطا می‌شود، هرچند ممکن است تعداد تطبیق‌های نهایی کاهش یابد.

Homography

Homography یک مفهوم بنیادی در بینایی ماشین و هندسه تصویری است که به کمک آن می‌توان رابطه‌ی هندسی بین دو تصویر از یک صحنه را مدل‌سازی کرد. به طور خاص، Homography یک **ماتریس ۳×۳** است که هر نقطه از تصویر اول را به نقطه متناظر در تصویر دوم نگاشت می‌کند، در حالی که تغییرات شامل چرخش، مقیاس، جابجایی و پرسپکتیو را لحاظ می‌کند. این ماتریس امکان بازسازی هندسی و هم‌ترازی تصاویر را فراهم می‌آورد.

یکی از کاربردهای مهم Homography در تطبیق تصاویر است. هنگامی که دو تصویر از یک شیء یا صحنه با زاویه دید متفاوت گرفته می‌شوند، نقاط کلیدی مشابه بین دو تصویر شناسایی می‌شوند. با محاسبهHomography ، می‌توان موقعیت دقیق هر نقطه از تصویر اول را روی تصویر دوم تعیین کرد. این امر به ویژه در کاربردهایی مانند شناسایی اشیا، تشخیص حرکت، و ردیابی در ویدئوها کاربرد دارد.

محاسبه Homography معمولاً با استفاده از نقاط کلیدی و توصیف‌گرهای استخراج‌شده از تصاویر انجام می‌شود. ابتدا نقاط مشابه بین دو تصویر با الگوریتم‌هایی مانندSIFT ، SURFیا ORB پیدا می‌شوند. سپس با استفاده از روش‌هایی مانند **RANSAC**، یک ماتریس Homography تخمینی محاسبه می‌شود که بیشترین تعداد نقاط درست را نگه می‌دارد و نقاط نویزی یا اشتباه را حذف می‌کند. این فرآیند باعث می‌شود ماتریس نهایی مقاوم و قابل اعتماد باشد.

با داشتن ماتریسHomography ، می‌توان تصویر اول را روی تصویر دوم منطبق کرد، مختصات گوشه‌ها یا اشیای مورد نظر را انتقال داد و تصویر هم‌تراز ایجاد نمود. در عمل، این کار امکان رسم مستطیل روی تصویر دوم برای نشان دادن محل شیء یا هم‌ترازی پانوراماها را فراهم می‌آورد. به زبان ساده، Homographyمانند یک **نقشه‌ی هندسی** عمل می‌کند که نقاط تصویر اول را به موقعیت صحیح خود روی تصویر دوم هدایت می‌کند.

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

img1 = cv2.imread('images/100tomani.jpg', cv2.IMREAD_GRAYSCALE)
img2_color = cv2.imread('images/eskenas.jpg')
img2 = cv2.cvtColor(img2_color, cv2.COLOR_BGR2GRAY)

sift = cv2.SIFT_create()
```

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

bf = cv2.BFMatcher(cv2.NORM_L2)
matches = bf.knnMatch(des1, des2, k=2)
good = [m for m, n in matches if m.distance < 0.7 * n.distance]

result = img2_color.copy()
if good:
    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 = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)
    h, w = img1.shape
    dst = cv2.perspectiveTransform(np.float32([[0,0],[0,h-1],[w-1,h-1],[w-1,0]]).reshape(-
1,1,2), M)
    result = cv2.polylines(result, [np.int32(dst)], True, (0,255,0), 3, cv2.LINE_AA)

plt.figure(figsize=[15,4])
plt.subplot(131); plt.imshow(cv2.cvtColor(cv2.imread('images/100tomani.jpg'),
cv2.COLOR_BGR2RGB)); plt.title('Image1')
plt.subplot(132); plt.imshow(cv2.cvtColor(img2_color, cv2.COLOR_BGR2RGB));
plt.title('Image2')
plt.subplot(133); plt.imshow(cv2.cvtColor(result, cv2.COLOR_BGR2RGB)); plt.title('Result')
plt.show()
```

در این کد:

```
bf = cv2.BFMatcher(cv2.NORM_L2)
matches = bf.knnMatch(des1, des2, k=2)
good = [m for m, n in matches if m.distance < 0.7 * n.distance]
```

BFMatcher یا Brute-Force Matcher یک الگوریتم ساده و مستقیم برای پیدا کردن نقاط مشابه بین دو تصویر است.

cv2.NORM_L2 مشخص می‌کند که برای مقایسه Descriptorهای عددی (مثل SIFT یا SURF) از فاصله اقلیدسی (Euclidean Distance) استفاده شود.

knnMatch برای هر Descriptor در تصویر اول، دو Descriptor نزدیک‌ترین تصویر دوم را پیدا می‌کند (k=2).

```
good = [m for m, n in matches if m.distance < 0.7 * n.distance]
```

این خط **بهترین تطبیق‌ها را انتخاب می‌کند**.

شرط `m.distance < 0.7 * n.distance` می‌گوید: اگر فاصله بهترین match (m) کمتر از ۰.۷ برابر فاصله دومین match (n) باشد، آن تطبیق قابل اعتماد است.

این کار باعث حذف تطبیق‌های اشتباه و نویزی می‌شود.

```
if good:
    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)
```

هر عنصر در `good` یک match است که شامل دو index می‌باشد:

- `queryIdx`: index نقطه کلیدی تصویر اول
- `trainIdx`: index نقطه کلیدی تصویر دوم

`[kp1[m.queryIdx].pt for m in good]` لیستی از مختصات `(x, y)` نقاط کلیدی تصویر اول می‌سازد.

`[kp2[m.trainIdx].pt for m in good]` مختصات متناظر نقاط کلیدی در تصویر دوم را می‌سازد.

`reshape(-1,1,2)` قالب داده را به شکل مورد نیاز `cv2.findHomography` تبدیل می‌کند (لیست نقاط با ابعاد صحیح).

```
M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)
```

`cv2.findHomography` یک **ماتریس ۳×۳** محاسبه می‌کند که نقاط تصویر اول را به تصویر دوم نگاشت می‌کند.

`RANSAC` یک الگوریتم مقاوم در برابر نقاط نویزی است و نقاط اشتباه یا outlier را نادیده می‌گیرد.

•خروجی:

- `M` ماتریس Homography
- `Mask` مشخص می‌کند کدام تطبیق‌ها در محاسبه نهایی لحاظ شده‌اند.

- مدرس محمد کهکشانی (مدرس رسمی دانشگاه هاروارد)


```
h, w = img1.shape
    dst = cv2.perspectiveTransform(np.float32([[0,0],[0,h-1],[w-1,h-1],[w-1,0]]).reshape(-1,1,2), M)
```

- `[[0,0],[0,h-1],[w-1,h-1],[w-1,0]]` مختصات گوشه‌های تصویر اول (بالا-چپ، پایین-چپ، پایین-راست، بالا-راست).
- `perspectiveTransform` گوشه‌های تصویر اول را با ماتریس Homography به تصویر دوم می‌برد، یعنی **موقعیت واقعی تصویر اول روی تصویر دوم را محاسبه می‌کند**.