Image Feature Matching

17010826 김성민

CONTENTS

CONTENTS

Image Feature Matching

목표 : Image Feature Extraction 단계에서 얻은 Image Feature를 Descriptor를 이용해 매칭하는 것

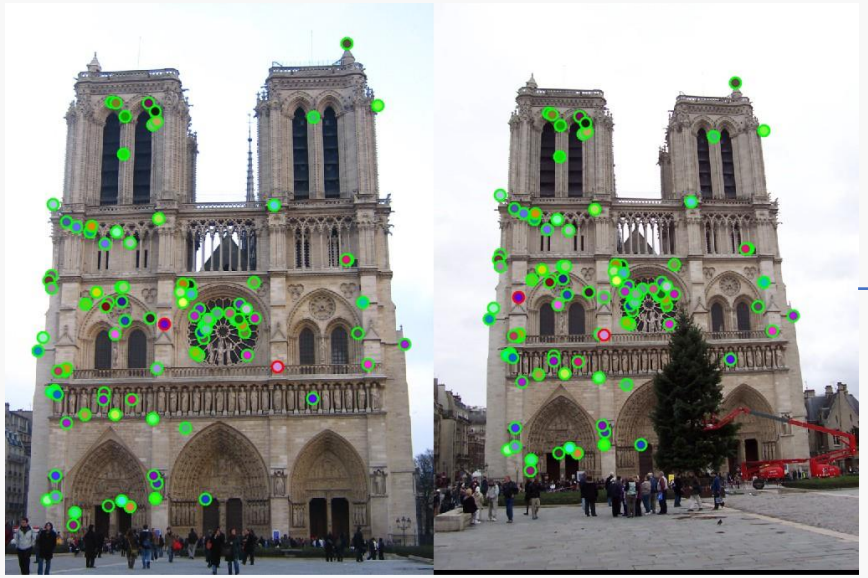
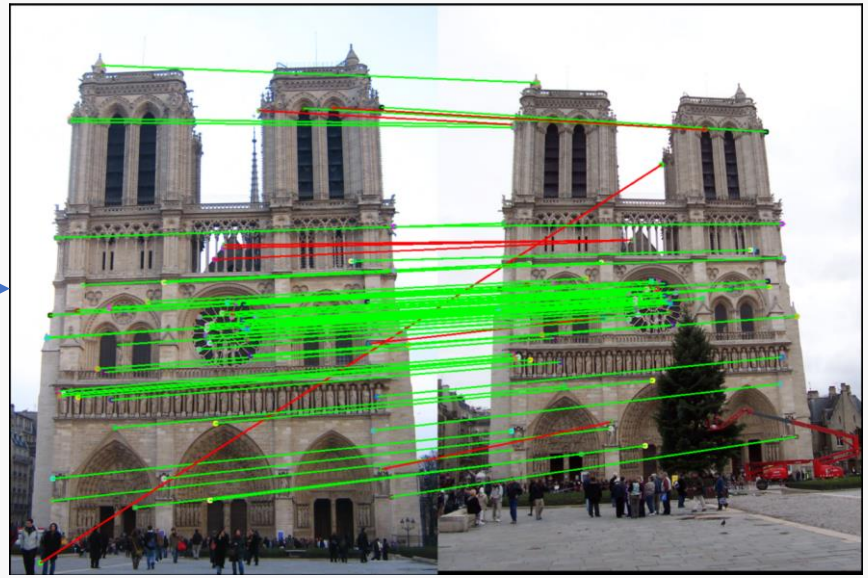


Image Feature Extraction 단계에서 얻은 Image Feature



각 이미지에서 같은 부분을 표현하는 Feature끼리 Matching

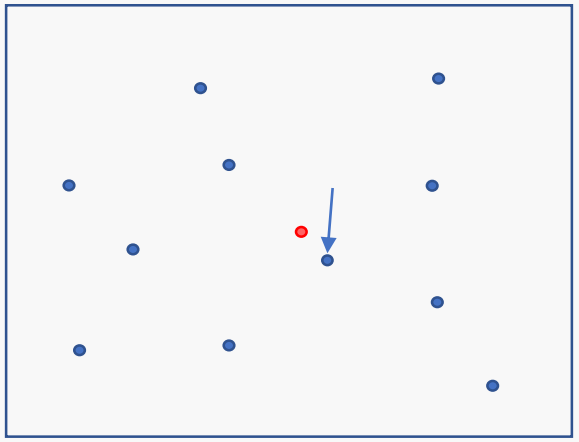
초록색 선은 잘 Matching된 Feature 쌍을,
빨간색 선은 잘 Matching되지 못한 Feature 쌍을 나타낸다.

Image Feature Matching

01 Nearest Neighbor Search

Nearest Neighbor란?

Descriptor vector의 Euclidean 거리가 가장 가까운(Closest) keypoint



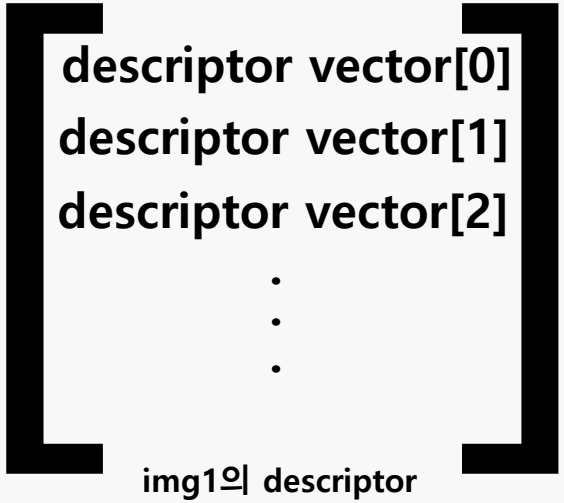
옆의 파란 상자가 벡터 공간, 점들을 벡터라고 할 때,
빨간 벡터 입장에서 가장 가까운 벡터는 어떤 벡터일까?

바로 화살표가 가리키는 벡터가 빨간 벡터와 가장 가까운(Closest) 벡터이다.

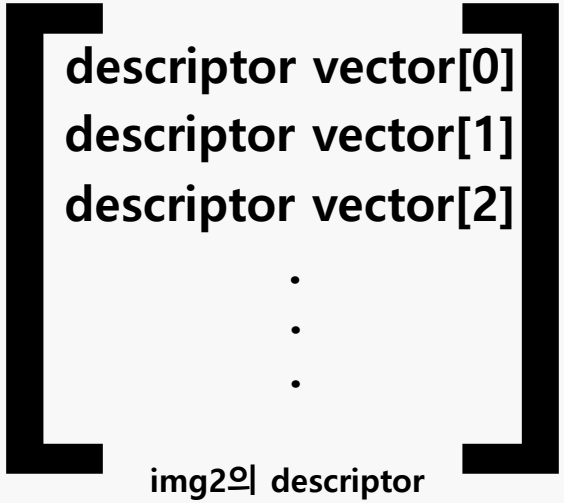
⟶ 빨간 벡터 입장의 Nearest Neighbor

Euclidean 거리 공식 : $d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$

Feature Matching에서 Nearest Neighbor



img1의 descriptor



img2의 descriptor

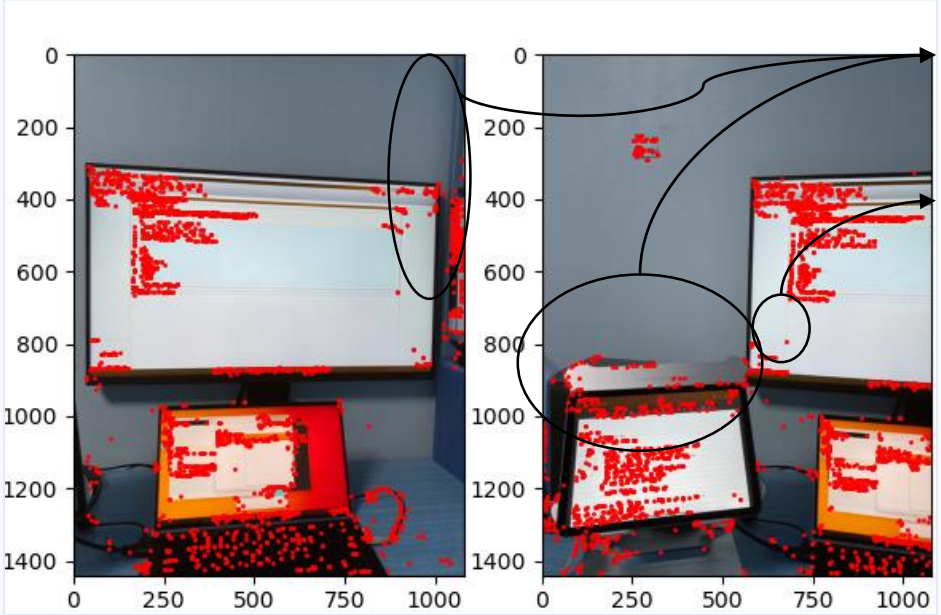
img1의 descriptor vector[0]에서
img2의 모든 descriptor vector와 거리를 비교해
Nearest neighbor를 찾는다.

img1의 다른 모든 descriptor vector에 대해서도
img2의 모든 descriptor vector와 거리를 비교해
Nearest neighbor를 찾는다.

Image Feature Matching

02 Nearest Neighbor Distance Ratio (NNDR)

Nearest Neighbor와 바로 매칭해도 될까?



두 이미지가 겹치지 않는 부분에서도 Feature들이 존재한다.

혹은 겹치더라도 올바르게 매칭될 Feature가 없는 경우가 존재한다.

(noise로 예상되는 부분에 Feature가 존재하거나 한 쪽 이미지에서만 Feature가 추출된 경우)

따라서 모든 Feature를 Nearest Neighbor와 연결하면 안된다.

→ Nearest Neighbor가 무조건 객체의 같은 부분을 나타내는 Feature가 아니기 때문

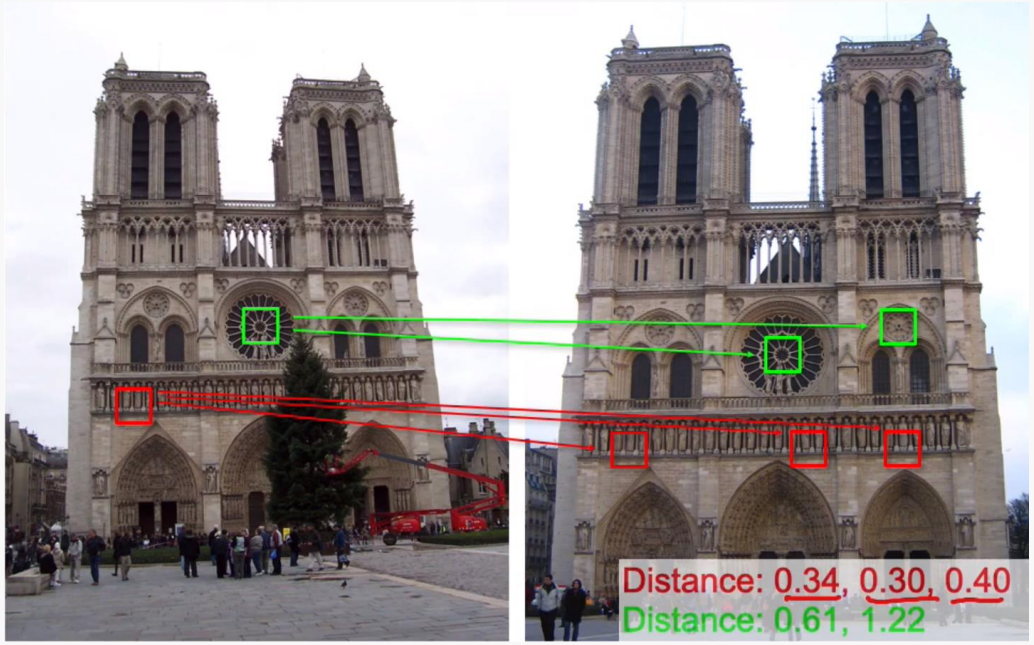
→ 서로 상관없는 Feature끼리 서로 연결된다.

좋은 Matching을 이룰 수 없는 Feature들은 Matching하면 안되며 이를 구분할 방법이 필요하다.

Nearest Neighbor Distance Ratio (NNDR)

좋은 Matching 쌍을 구분하기 위한 방법

가장 가까운 descriptor vector의 거리와 두 번째로 가까운 descriptor vector의 거리를 비교한다.



한편, Nearest Neighbor와의 Distance의 크기 자체는 Matching하기에 좋은 Feature 쌍인지 구분하기가 어렵다는 사실을 알 수 있다.

왼쪽의 Feature들을 오른쪽과 Matching 시키기 위해 가장 가까운 Feature와 그 다음으로 가까운 Feature들을 살펴보았다.

초록색 Feature의 경우 Matching 후보들 중에 어디에 Matching 시켜야 할 지가 비교적 명확하며 두 후보의 Distance의 차가 크다.

빨간색 Feature의 경우 Matching 후보를 3개로 늘렸음에도 Feature 부분만 보았을 때는 어디에 Matching 시켜야 할 지가 비교적 불명확하며 세 후보의 Distance가 비슷하다.

좋은 Matching을 하기 위해서는
Feature의 Closest Neighbor와 Second-Closest Neighbor와의 Distance를 비교해서 Matching에 적절한 Feature 쌍을 찾아야 한다.

Nearest Neighbor Distance Ratio (NNDR)

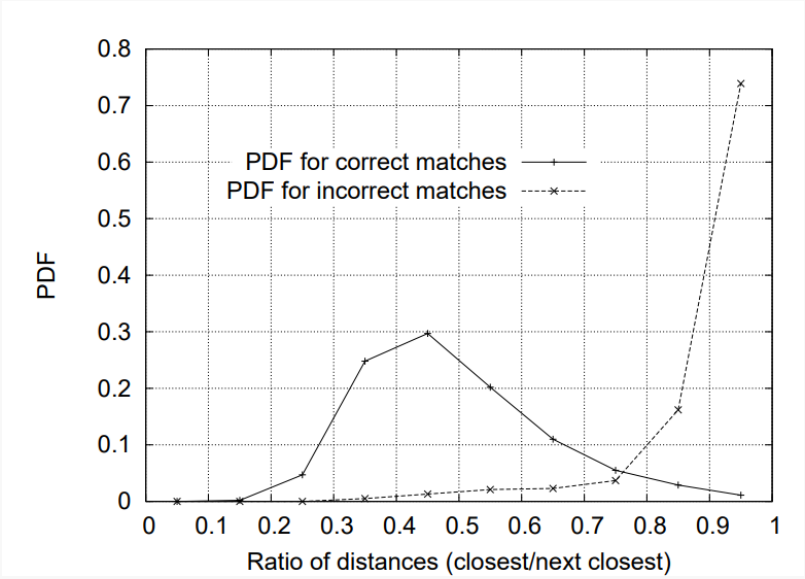
Closest Neighbor와의 거리를 NN1, Second-Closest Neighbor와의 거리를 NN2라고 한다면

만약, NN1과 NN2의 값이 비슷하다면 $\frac{NN1}{NN2} \approx 1$ 이고,

NN1과 NN2의 값이 비슷하지 않다면 $\frac{NN1}{NN2}$ 는 0과 가까워진다.

➡ 낮은 $\frac{NN1}{NN2}$값을 사용하는 것이 좋다.

그렇다면 $\frac{NN1}{NN2}$의 기준점을 어떻게 설정하는 것이 좋을까?



40000개의 keypoint에 대해 사람이 직접 좋은 matching쌍인지 아닌지 직접 판단해서 $\frac{NN1}{NN2}$에 대한 확률 밀도 함수를 구함

PDF(Probability Density Function) : 확률 밀도 함수

$\frac{NN1}{NN2}$의 값이 낮으면 낮을수록 incorrect matching의 수가 줄어들지만 correct matching의 수도 줄어듦

반면, $\frac{NN1}{NN2}$의 값이 높을수록 correct matching의 수가 늘어나지만 incorrect matching의 수도 늘어남

올바른 matching 쌍의 경우 그래프가 비교적 종 모양을 보이지만
올바르지 않은 matching 쌍의 경우 약 0.75를 기준으로 그래프가 급격히 증가

➡ 즉, 0.7~0.8 사이의 값을 $\frac{NN1}{NN2}$의 threshold로 설정하는 것이 바람직하다.

SIFT 논문의 저자는 $\frac{NN1}{NN2}$이 0.8일 경우 90%의 잘못된 Matching 쌍을 걸러낼 수 있으며 버려지는 올바른 Matching 쌍은 5%라고 밝혔다.

Image Feature Matching

03 프로그램 순서도

Image Feature Matching에 필요한 입력값

Image Feature Extraction에서 얻은 keypoint와 descriptor

```
In [7]: print("키포인트의 타입 : ", type(kp1))
        print("키포인트의 길이 : ", len(kp1))
        print("개별 키포인트의 타입 : ", type(kp1[0]))
        print("키포인트의 위치 예시 : ", kp1[0].pt)
        print("descriptor의 타입 : ", type(des1))
        print("descriptor의 크기 : ", des1.shape)
        print("descriptor 예시 : ", des1[0])

키포인트의 타입 :  <class 'tuple'>
키포인트의 길이 :  2002
개별 키포인트의 타입 :  <class 'cv2.KeyPoint'>
키포인트의 위치 예시 :  (21.07024383544922, 1090.223388671875)
descriptor의 타입 :  <class 'numpy.ndarray'>
descriptor의 크기 :  (2002, 128)
descriptor 예시 :  [136.  12.   0.   0.   0.   0.   0.   0. 178.  17.   0.   0.   0.   0.
   0.   1.  97.   4.   0.   0.   1.   1.   0.   1.   0.   1.   1.   1.
   2.   1.   0.   0. 159.  13.   0.   0.   0.   0.   0.   9. 178.   7.
   0.   0.   0.   0.   0.  15. 120.   2.   0.   0.   0.   0.   0.   6.
   0.   0.   0.   1.   1.   2.   3.   1. 160.   3.   0.   0.   0.   0.
   0.  25. 178.   3.   0.   0.   0.   0.   0.  16. 117.   6.   0.   0.
   0.   0.   0.   3.   3.   1.   0.   1.   1.   1.   2.   2. 134.   6.
   0.   0.   0.   0.   0.   3. 178.  16.   0.   0.   0.   0.   0.   1.
  92.   6.   0.   0.   0.   0.   0.   1.   2.   2.   1.   2.   1.   0.
   0.   0.]
```

keypoint들은 tuple형태로 저장되며
개별 keypoint는 openCV의 KeyPoint Class 형태로 저장되어 있음

Descriptor들은 numpy.ndarray 형식으로
2차원 행렬 구조로 저장되어 있음

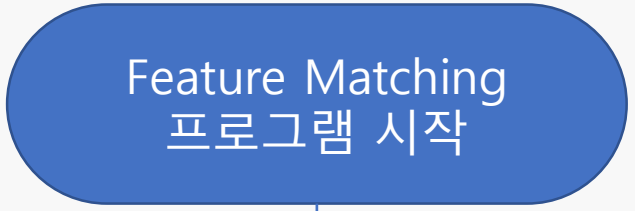
→ 행의 수는 keypoint의 개수와 같으며 각 행이 keypoint의 descriptor를 나타내는 vector로 이루어져 있음

→ SIFT에서 얻은 descriptor vector의 차원은 128이므로 descriptor를 표현하는 numpy.ndarray는 128개의 column을 가짐

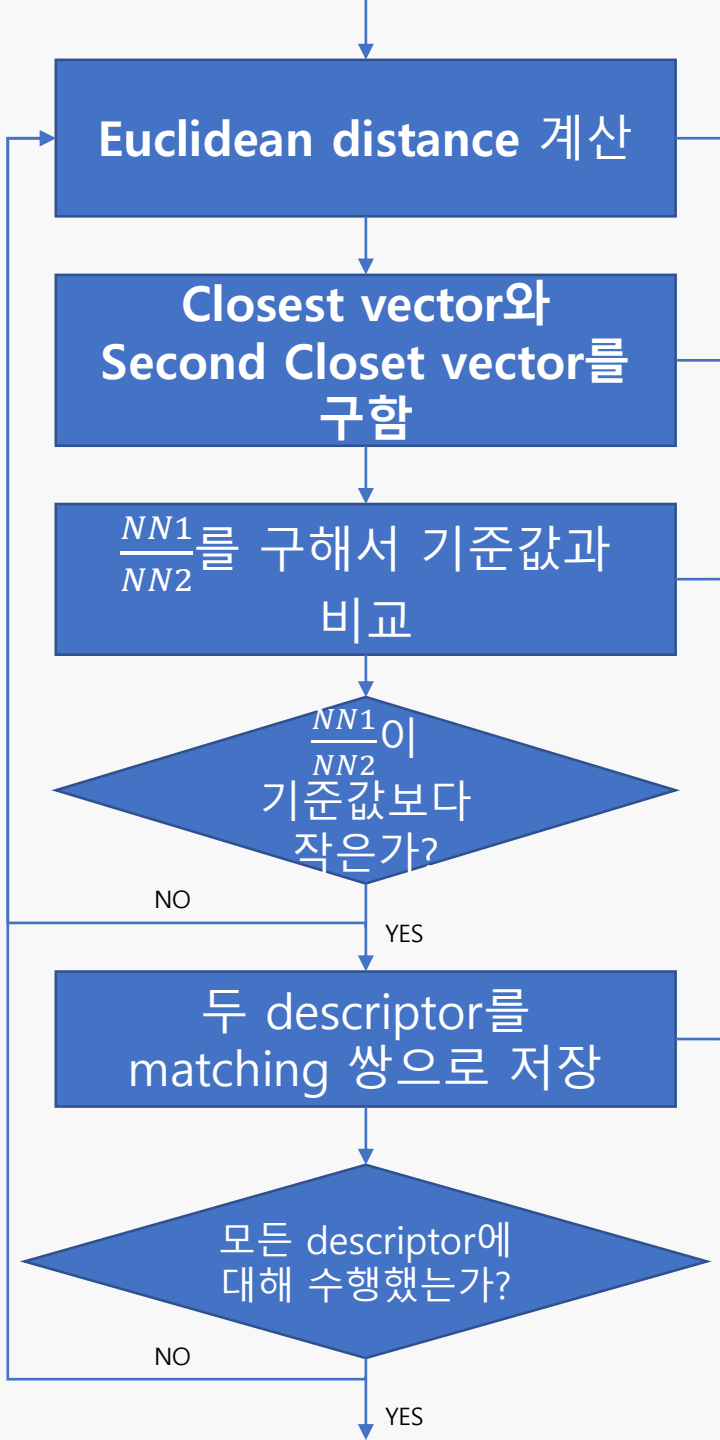
Decriptor의 index번째 행이 keypoint들이 저장된 tuple의 index번째 keypoint를 설명

프로그램 순서도

https://www.youtube.com/watch?v=rjdE5Kx5sEc 참고하여 작성한 코드



SIFT 후 얻은 두 쌍의
keypoint와 descriptor vector로 이루어진 2차원 list를
입력 받음



한 이미지의 각각의 descriptor에서
다른 이미지의 모든 descriptor와의 거리를 계산한다.

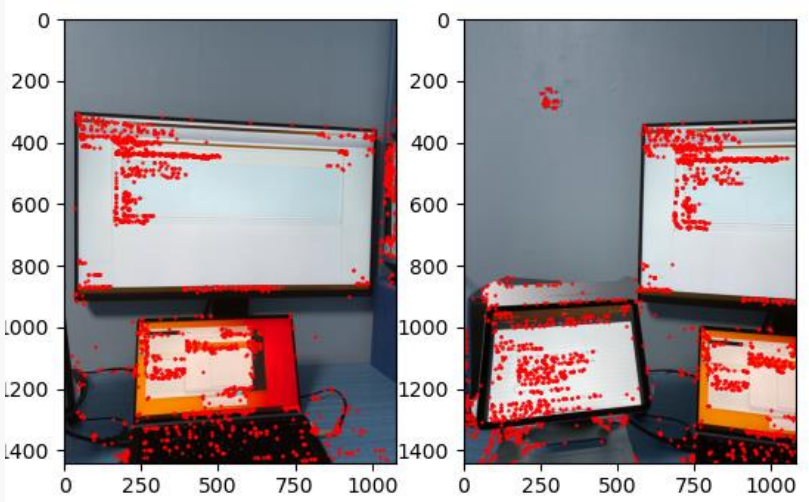
Euclidean distance가 가장 가까운 descriptor 벡터와
두 번째로 가까운 descriptor 벡터를 찾는다.

기준값을 [0.7, 0.8]에서 설정

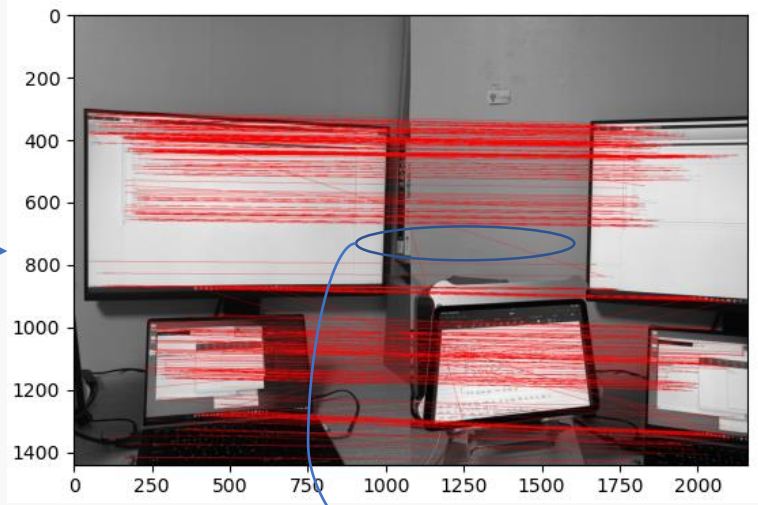
Euclidean distance 계산에서 Nearest Neighbor를
이루는 두 descriptor

저장된 matching 쌍을 이용해 Feature Matching

Euclidean distance 계산에서 Nearest Neighbor를 이루는 두 descriptor



Matching



Threshold=0.7

잘못된 Matching쌍 존재

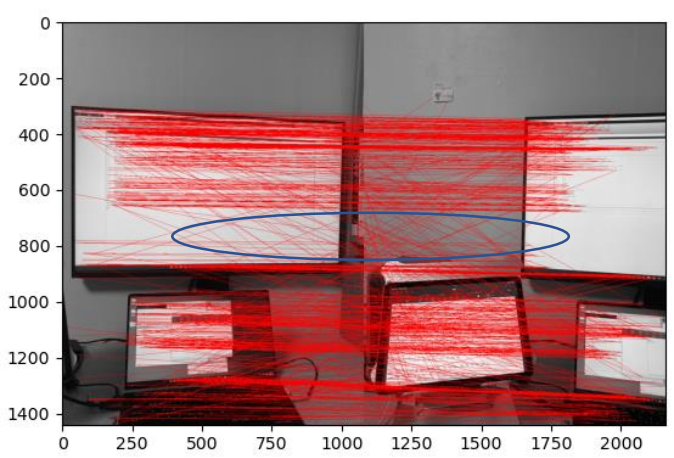
한편, Image Feature Matching 프로그램의 출력값으로 Matching된 Feature 쌍을 출력해 Image Stitching의 다음 단계에서 사용할 수 있다.

Image Feature Matching

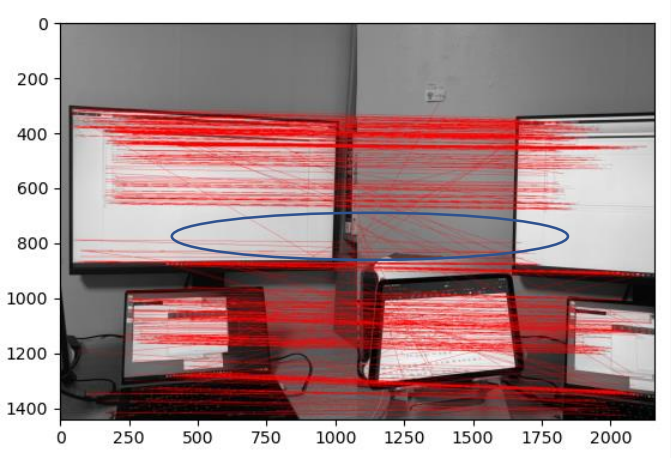
04 NNDR에 따른 Matching 결과

$\frac{NN1}{NN2}$에 따른 Matching 결과

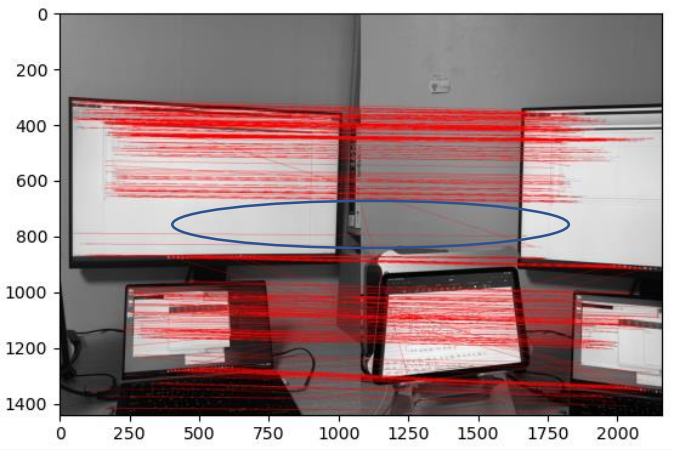
$\frac{NN1}{NN2}$ = 0.9



$\frac{NN1}{NN2}$ = 0.8



$\frac{NN1}{NN2}$ = 0.7



$\frac{NN1}{NN2}$가 0.9인 경우는 $\frac{NN1}{NN2}$가 0.8과 0.7인 경우에 비해

잘못 매칭된 것으로 보이는 Matching 쌍의 개수가 눈에 띄게 많다.

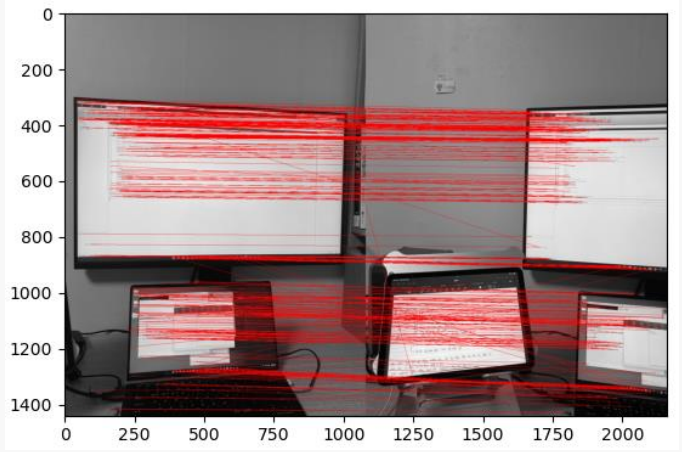
두 사진을 높낮이 변화가 거의 없게 좌우로만 카메라 방향을 변화시켜 찍은 사진이므로

그림의 Matching 쌍에서 기울기가 x축과 차이가 많이 나는 경우는 잘못 Matching 되었다고 생각할 수 있다.

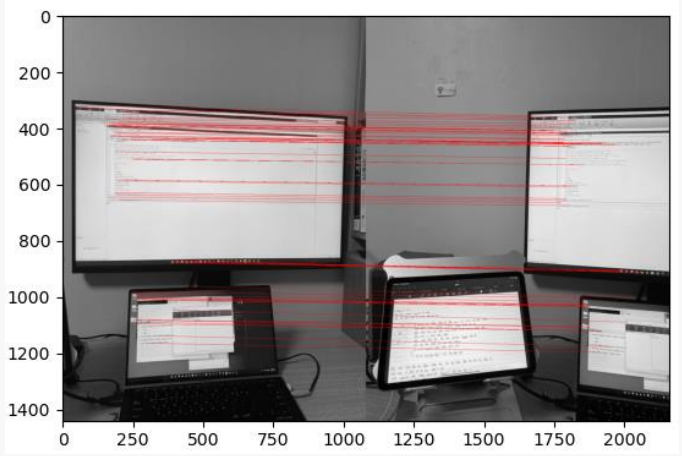
→ $\frac{NN1}{NN2}$이 0.8을 넘어가면 잘못된 Matching 쌍이 급격하게 늘어난다는 점을 눈으로 확인할 수 있다.

$\frac{NN1}{NN2}$에 따른 Matching 결과

$\frac{NN1}{NN2} = 0.5$



$\frac{NN1}{NN2} = 0.3$



$\frac{NN1}{NN2}$가 0.7보다 작아지게 되면 잘못된 Matching 쌍의 수는

$\frac{NN1}{NN2}$에 비해 큰 차이가 없지만 잘 된 Matching의 수가 많이 줄어든다.

➡ 잘못된 Matching 쌍을 줄이기 위해 $\frac{NN1}{NN2}$를 지나치게 낮게 설정하면
잘 된 Matching 쌍을 많이 잃게 된다.

즉, 이론에서 $\frac{NN1}{NN2}$ 값이 [0.7, 0.8] 구간에 있어야 바람직하다는 사실을 실제 실험 결과에서도 확인할 수 있다.

THANK YOU –

감사합니다.