<모바일응용 소프트웨어 설계>

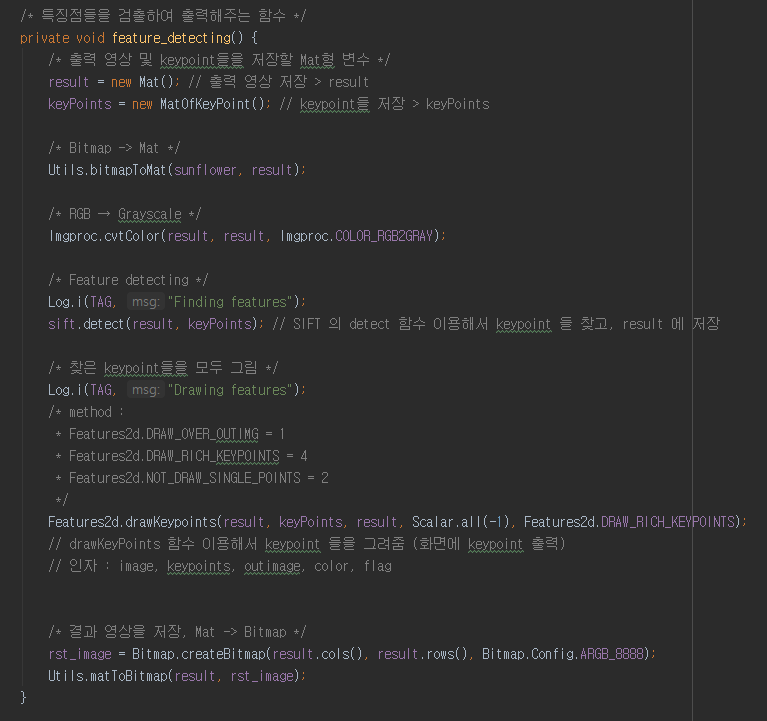
# 11주차 실습 과제

정보통신공학과

12161774 이 채 은

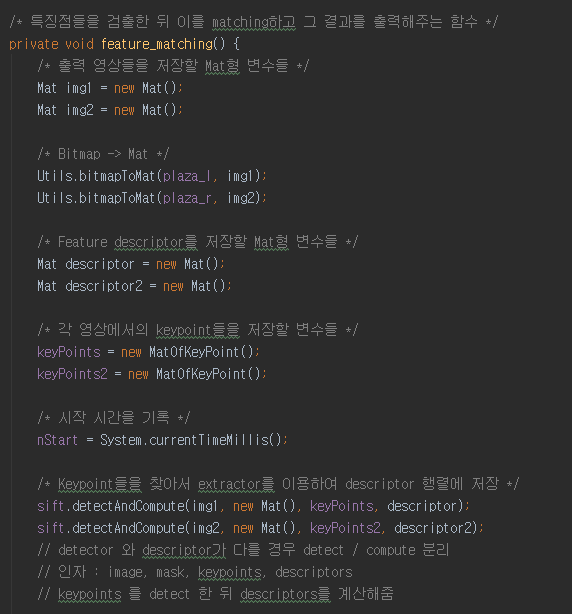
**[소스 코드]**

1. MainActivity.java : feature\_detecting( )



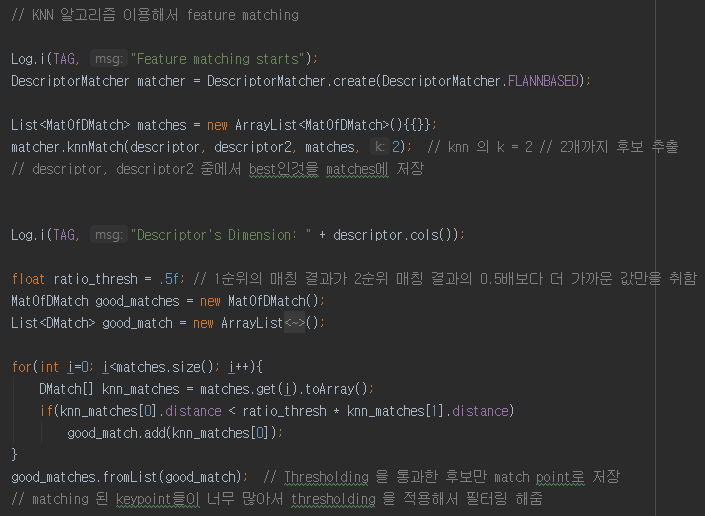
=> 특징점을 찾는데 이용되는 feature\_detecting 함수이다. MatOfKeyPoint 형식을 이용해서 추출한 특징점 keypoint를 저장한다. 먼저 bitmap 형식의 이미지를 mat 형식으로 저장하고, RGB 색을 Grayscaling 한다. 그 후 SIFT의 detect 함수를 이용해서 keypoint를 찾는다. 찾은 keypoint는 result에 저장한다. 따라서 Detect 가 끝나고 나면 특징점이 잡히고, 그 결과가 result에 저장되는 원리이다.

2. MainActivity.java : feature\_matching( )

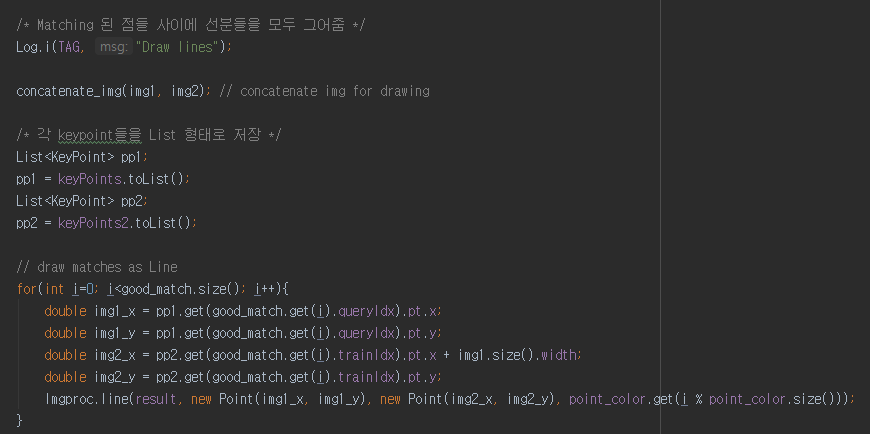


=> 검출한 특징점을 matching 하는 feature\_matching 함수이다.

두 개의 이미지를 mat 형식으로 저장할 mat 형 변수를 선언하고, bitmap 형식을 mat로 바꿔준다. 그리고 keypoint를 저장할 변수도 각각의 이미지에 대해서 MatOfKeyPoint 형으로 선언한다. Keypoint 들을 찾아서 descriptor를 계산해 descriptor 행렬에 저장하는 과정을 거쳐야 한다. Descriptor도 Mat 구조체를 사용한다. 이 때 Sift의 detectAndComute 를 이용한다. Detector 와 descriptor가 다른 경우에 이와 같이 detect 와 compute를 분리해서 따로 실행하고 합치는 과정이 있어야 하는데, 이를 한번에 해준다. 따라서 keypoint를 detect 한 뒤에 descriptor를 계산해주는 역할을 한다. detectAndCompute 함수가 끝나고 나면 keypoint 가 descriptor를 갖게 된다.

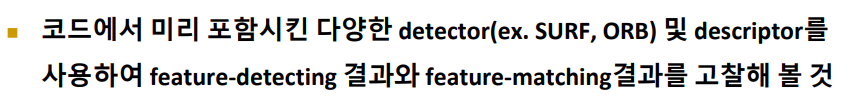


=> 찾은 특징점을 matching 하는데 사용되는 KNN 알고리즘에 대한 코드이다. Descriptor가 가장 가까운 k개를 뽑은 뒤에 distance를 계산해서 for문을 통과한 경우에만 good\_match에 저장하는 원리이다. 따라서 detecting 된 특징점을 filtering 해서 matching 하는 역할을 한다. for문에서는 모든 matches.size에 대해서 descriptor를 비교한다. 따라서 threshold 값에 따라서 거리가 더 가까운 경우에만 선택되어 for문의 조건을 통과하고 good\_match 에 저장되는 것이다. Keypoint의 값이 너무 많은 경우에 thresholding 을 적용해서 필터링을 하는 원리이다. 이 때 knn의 k 값은 선택할 후보의 개수인데, k=2이면 두 개의 후보를 속출해서 비교하겠다는 것이다.

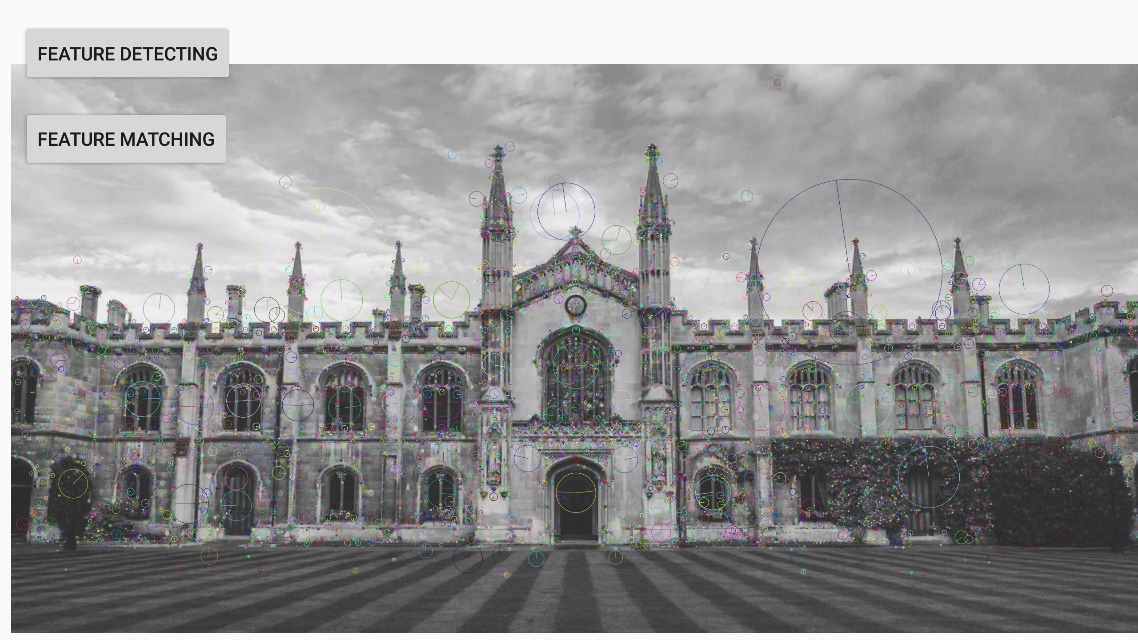


=> matching 된 점들끼리 선분을 이어서 matching 결과를 나타내기 위한 코드이다. for문에서 x, y 좌표를 구해와서 포인트를 찍고 line을 이용해서 선분으로 이어주는 원리이다. 이 때 두번째 이미지는 첫 번째 이미지 옆에 붙였기 때문에 두번째 이미지의 x 좌표는, 첫 번째 이미지의 width 만큼을 이동한다.

**[ 실습 결과 및 고찰 ]**

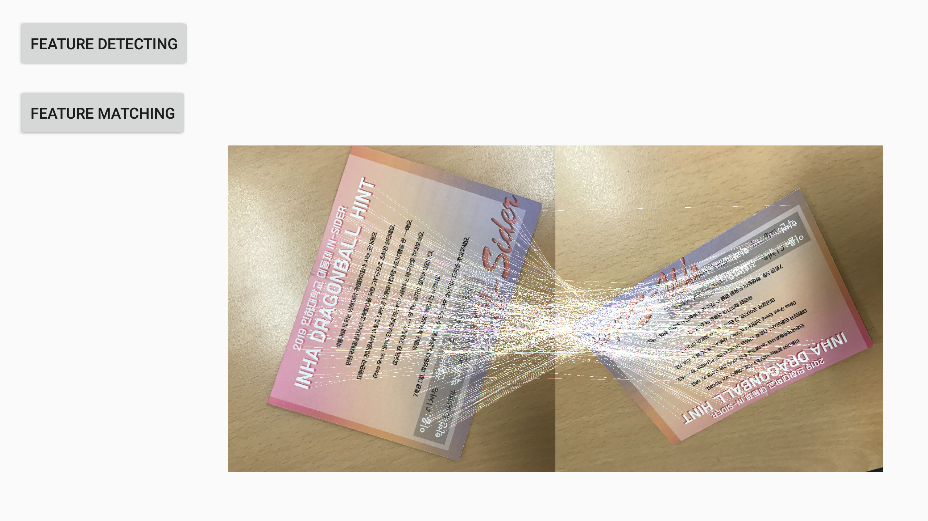




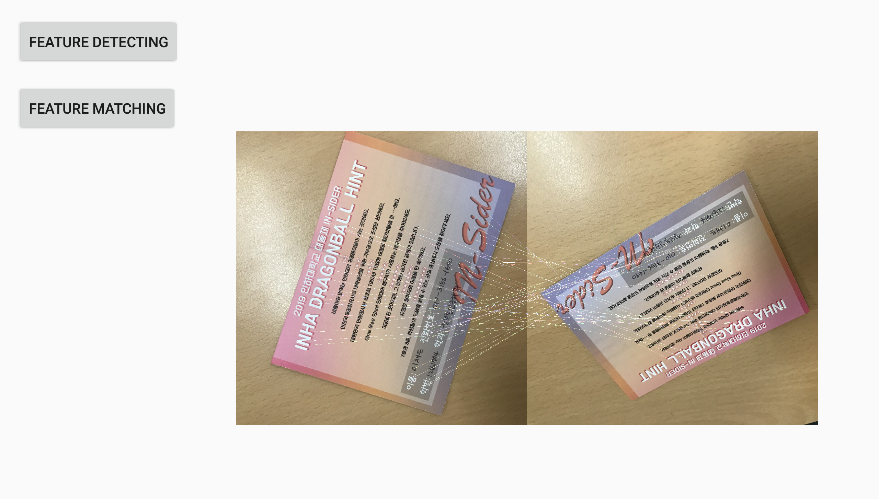
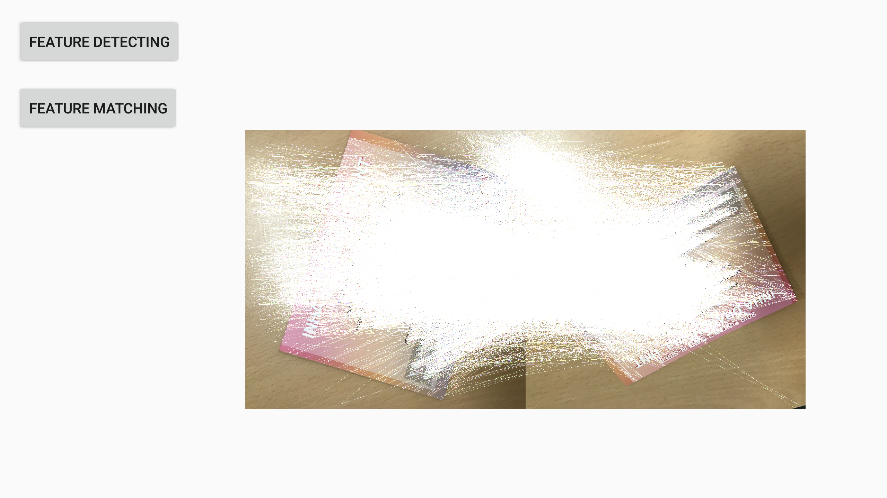


=> 이번 과제를 통해서 feature detecting 과 matching을 진행했다. 특징점을 찾는데 쓰이는 방법은 여러가지가 있다. 실습 시간에 진행했던 알고리즘은 SIFT이다. SIFT(Scale-Invariant Feature Transform) 알고리즘은 특징점을 추출하고, descriptor로 변환해주는 알고리즘이다. 이 알고리즘은 특히, scale(이미지 크기)이 변화되어도 특징점을 추출할 수 있다는 특징을 가지고 있다. OpenCV에서 특징은 다음과 같다. 가장 먼저 특징점을 찾아내고, 찾은 특징점은 extractor를 이용해 정규화한 descriptor로 저장한다. 따라서 descriptor의 거리를 계산해서 조건에 따라 맞는 조건일 경우에는 matching을 한다. 거리를 이용해 조건을 파악할 때 쓰이는 알고리즘은 KNN 알고리즘이다. KNN은 최근접 이웃 알고리즘으로, best 결과로 매칭된 경우와 second의 결과로 매칭된 경우를 비교해서 더 best인 경우를 찾아 최종 매칭 결과를 출력하는 방법이다. 따라서 매칭한 결과를 필터링할 수 있다. 따라서 threshold 값에 따라서 특징점을 matching 하는 선분의 개수가 달라진 것을 직접 확인할 수 있었다. 추가로 detector 방법에 SURF, ORB가 있다. SURF는 SIFT 연산의 복잡성을 해결하고, 계산 속도를 빠르게 한 방법이다. 하지만 SIFT 보다 정확성이 떨어지고, 그레이 공간 상의 정보만 이용하기 때문에 color 공간 상에 있는 정보들에 대해 구별하기 어렵다. 따라서 두개의 이미지로 비교할 때, 다른 한 장이 원래 이미지를 블러했거나 회전한 경우에는 특징점을 제대로 검출하고 매칭하지만, 다른 한 장이 원래 이미지의 뷰포인트와 다르거나 조명을 바꾼 경우에는 특징점을 틀리게 검출하고 매칭할 수 있다. ORB는 특징점 검출 시에 FAST를, 알고리즘 성능을 높이기 위해 BRIEF descriptor를 혼합해서 적용한 알고리즘을 사용한다. 이미지의 descriptor를 빠르게 계산하기 위해서 쓰이는 방법이 BRIEF이다. 따라서 코드 상에서는 ORB 객체를 초기화 하고, keypointer와 descriptor를 detectAndCompute를 사용해서 계산한다. SIFT를 이용해서 FEATURE DETECTING 한 결과는 위와 같았다. 특징점이 잡히고 원 모양으로 Blobs도 잡힌 것을 확인할 수 있었다.





=> SIFT FEATURE MATCHING 에서 ratio\_thresh 값을 0.3 / 0.5 / 1f 로 바꿔가면서 feature-matching 결과를 확인했다. 위의 사진은 기본으로 설정된 값 0.5f 일 때 이다. Feature detecting, matching 이 적당히 바르게 된 것을 확인했다.

=> ratio\_thresh 값을 0.3f 로 했을 때와, 1f 로 했을 때 결과는 순서대로 위의 사진과 같다.

Threshold 값을 너무 작게 하면, KNN 알고리즘에서 for문을 통과해서 조건에 부합한 good\_match에 저장되는 match point의 수가 적어진다. 따라서 matching 결과도 위와 같이 연결된 선분의 개수가 적은 것이다. 이 경우에는 구별해야 하는 특징점이 무시될 수 있는 단점이 있다. 반면에 Threshold 값을 너무 크게 하면, KNN 알고리즘에서 for문을 통과해서 good\_match에 저장되는 match point의 수가 많아진다. 따라서 matching 결과도 연결의 선분의 개수가 많게 되는 것이다. 이런 경우에는 제대로 된 특징점을 구별하기 어려워진다.