Automatic stitching of two images

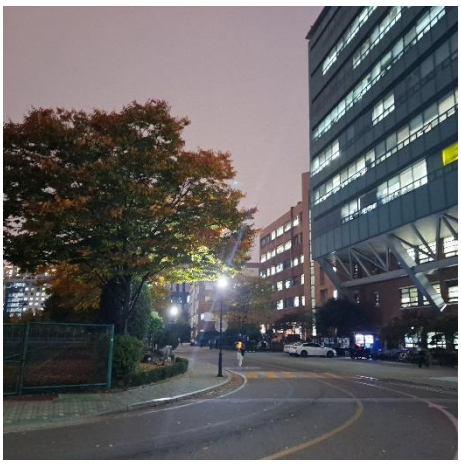
120230151 유지원

- 과제 목표 : 서로 다른 두 개의 이미지를 통해서 ***파노라마 이미지 만들기***
- 진행 과정 :

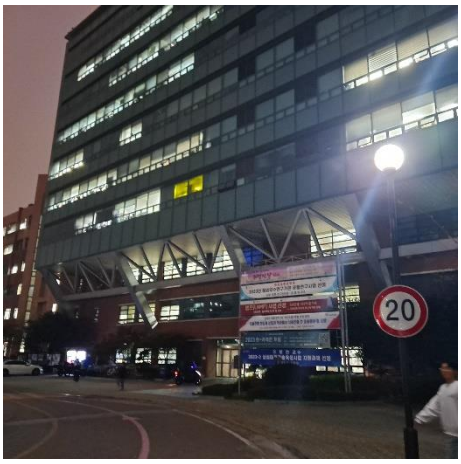
1) 서로 다른 두 개의 이미지 가져오기

<사진 1>

<사진 2>



<사진 3>



<사진 4>

위의 사진들은 휴대폰으로 촬영된 서강대학교 사진으로 파노라마를 만들 때는 위의 사진들을 전부 1024*1024 크기로 resize하여 사용하였다. 이때, <사진 1> 과 <사진 2>를 사용하여 <파노라마 1> 을 만들고 <사진 3> 과 <사진 4>를 사용하여 <파노라마 2>를 만들 것이다.

2) 파노라마 만들기 위한 Code

A. 입력 받은 이미지를 grayscale로 변환

: '**cv2.cvtColor()**' 함수를 사용하여 RGB 이미지를 gray 이미지로 변환. 이는 이미지 특징을 추출하는 데 큰 도움을 얻을 수 있음.

B. ORB keypoints 와 descriptors 검출

: '**cv2.ORB_create()**' 함수를 사용하여 ORB 특징 검출기를 생성. 이후 ORB를 사용하여 각 이미지에서 keypoints 와 descriptors 추출

**ORB keypoints 란 Oriented FAST and Rotated BRIEF keypoints로 이미지의 특징점을 나타냄. 특징점이란 이미지 내의 고유하거나 주요한 위치를 말하는 것으로 이를 통해 이미지 매칭을 가능하게 함. 이때 추출한 descriptors는 특징점 주변의 형태와 구조를 나타내며 두 이미지 간의 특징점 매칭하는데 필요함.*

C. BruteForce Matcher

: **'cv2.BFMatcher()'** 함수를 사용하여 두 이미지의 descriptor 간 거리를 계산하여 최적의 매칭을 찾음. 즉, 두 이미지 간의 매칭을 수행하기 위해서 Bruteforce 매칭 방법을 사용함. 이때 **'cv2.NORM_HAMMING'**은 descriptor 간 거리 측정 방법을 나타냄.

D. 매칭 정렬 및 Good matches의 위치 추출

: 매칭된 keypoints들을 거리에 따라 정렬. 이중 거리가 가까운 매칭이 좋은 매칭으로 간주됨. 따라서 매칭된 keypoints들 중 가장 거리가 짧은 것들을 사용하여 두 이미지 간의 관련성을 확인하고, 이러한 keypoints 사용하여 파노라마 이미지 생성.

E. Homography 계산 및 이미지 변환

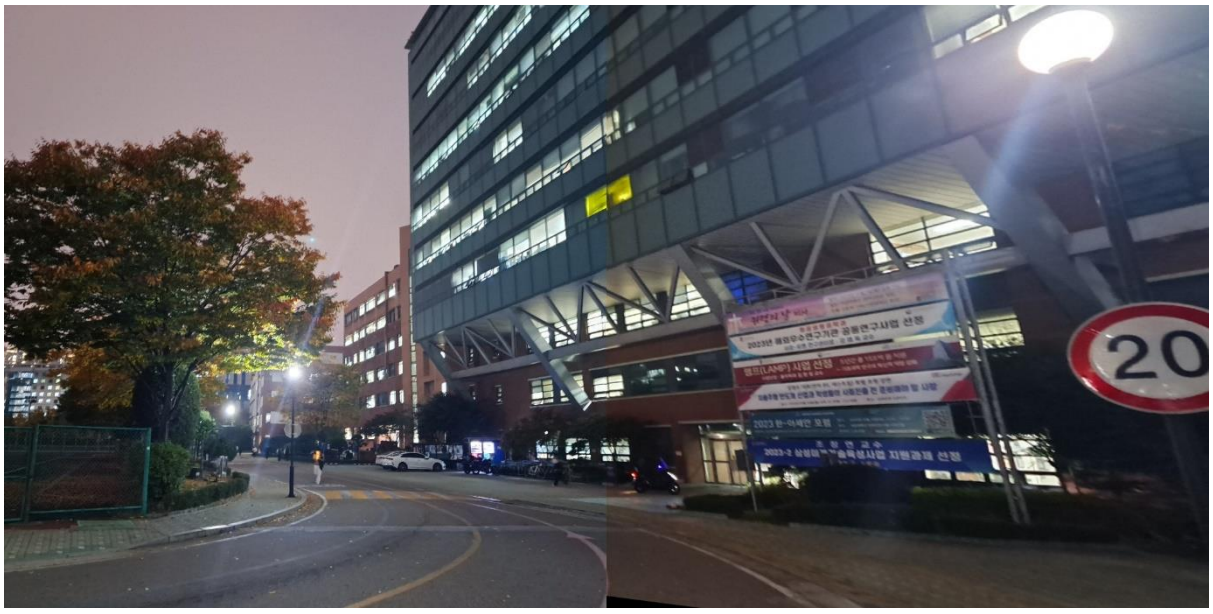
: **'cv2.findHomography'** 함수를 사용하여 **Good matches** 으로부터 Homography 행렬을 계산. Homography 행렬은 하나의 이미지를 다른 이미지에 정렬하는 변환 정보를 제공. 이때, Homography 행렬을 추정할 때 outliers 존재할 수 있어 RANSAC 알고리즘 사용하여 outliers에 강건한 추정 수행. 이후 **'cv2.warpPerspective'** 함수 사용하여 Homography 행렬을 통해 두 이미지를 정렬하고 크기 조정하여 최종 파노라마 이미지 생성.

```python
def stitch_images(img1, img2):
    # Convert images to grayscale
    gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
    gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)

    # Detect ORB keypoints and descriptors
    orb = cv2.ORB_create()
    keypoints1, descriptors1 = orb.detectAndCompute(gray1, None)
    keypoints2, descriptors2 = orb.detectAndCompute(gray2, None)

    # Use BruteForce Matcher
    bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
    matches = bf.match(descriptors1, descriptors2)

    # Sort matches based on their distances
    matches = sorted(matches, key = lambda x:x.distance)

    # Extract location of good matches
    points1 = np.zeros((len(matches), 2), dtype=np.float32)
    points2 = np.zeros((len(matches), 2), dtype=np.float32)

    for i, match in enumerate(matches):
        points1[i, :] = keypoints1[match.queryIdx].pt
        points2[i, :] = keypoints2[match.trainIdx].pt

    # Find homography
    h, mask = cv2.findHomography(points1, points2, cv2.RANSAC)

    # Use this matrix to transform the images
    height, width, channels = img2.shape
    panorama = cv2.warpPerspective(img1, h, (width * 2, height))
    panorama[0:height, 0:width] = img2

    return panorama
```

<그림 1. Code>

3) 최종 파노라마 사진

<사진 5. 사진 1 + 2 파노라마>



<사진 6. 사진 3 + 4 파노라마>