HOMEWORK 1

IMAGE MATCHING

Le Van Hung (黎文雄)

Student ID: 0860831

1. Step by Step to implement SIFT and Matching

a. Scale-space peak selection

- Generate all gaussian images: change the size of the image to double size and use this changed image as the first image in the pyramid → using cv2.GaussianBlur() to create other images base on corresponding sigma, reduce two times the size of the image after each octave using cv2.resize(). See function named gen_all_image(original_image) for more detail.

- Generate all DoG images: subtract two each gaussian images in each octave to get DoG image. See function named gen_all_DoG(all_image) for more detail.

- Find candidate keypoint: each pixel in DoG we have 26 neighbors, this pixel is one of the candidates keypoint if it is minimum or maximum of total 27 pixel. See function extrema_detection() for more detail.

- parameter setting: TOTAL_OCTAVE = 4, TOTAL_IMAGE_EACH_OCTAVE = 5, SIGMA = 1.6, K = sqrt(2), TOTAL_DOG_EACH_OCTAVE = 4.

b. Keypoint localization

- With each keypoint, we have 3x3x3 cube, using function localize_keypoint(rubik3x3) to calculate the first derivate, the second derivate matrix, offset, constrast threshold, trace and det of the center point of 3x3x3 cube.
- Eleminating "weak" keypoints using this condition:

  - Reject flats: $D(\hat{\mathbf{x}}) < 0.03$
  - Reject edges: $r < 10$

- Repeat this in all pixels in 1st and 2nd DoG layer of each octave. With each found keypoint, convert it to cv2.KeyPoint() object. See function find_one_keypoint() for more detail.
- Parameter setting: CONTRAST_THRESHOLD = 0.04, EIGENVALUE_RATIO (R) = 10, BORDER_WIDTH = 5 (ignore some pixel near boder, it releated to create descriptor)

c. Orientation Assignment

- Firstly, calculate the size of region around keypoint to calculate gradient based on SCALE_FACTOR and RADIUS_FACTOR. Using fuction named calculate_magnitude_orientation() to calculate gradient magnatude and gradient orientation in each point.
- Create a histogram, in this histogram, the 360 degrees are broken into 36 bins (each 10 degree), put the keypoint with corresponding orientation to 36 bins. After that, finding highest peak and peaks higher than 0.8*highest peak. See function keypoint_orientation() and find_all_keypoints() for more detail.
- Parameter setting: GOOD_PEAK_RATIO = 0.8, RADIUS_FACTOR = 3, SCALE_FACTOR = 1.5, TOTAL_BIN = 36, BIN_WIDTH = 10.

d. Keypoint descriptor

- Using unpackOctave(keypoint) function to get keypoint parameter from cv2.KeyPoint() object.
- In each keypoint, we calculate square neighborhood around each keypoint base on scale_multiplier, keypoint.scale, keypoint.size and window_width. After that, calculating gradient magnitude and gradient orientation like we did when computing keypoint orientations. However, instead of actually building a histogram and accumulating values, we simply store the histogram bin index and bin value for each pixel. Note that here our histograms have only 8 bins to cover 360 degrees, instead of 36 bins as before.
- Create histogram_tensor(row,col,orientation_bin) = magnitude to save magnitude value, using “the formulas and a good visualization” to resolve problem of not-integer value index of row, col. After that flat histogram_tensor to vector and append each this vector to descriptor array. See function generateDesscriptors() for more detail.
- Parameter setting: window_width = 4, num_bins = 8, scale_multiplier = 8

e. Keypoint Matching

- I use FLANN to find all matching. We perform approximate nearest neighbors search on the two sets of descriptors to find similar keypoints. Keypoint pairs closer than a threshold are considered good matches. Finally, we perform RANSAC on the keypoint matches to compute the best fit homography. See function plot_match_image() for more detail.
- Parameter setting: FLANN_INDEX_KDTREE = 0, the ratio of closest-distance = 0.7

2. Plot all 4 sets of images

a. mountain Set

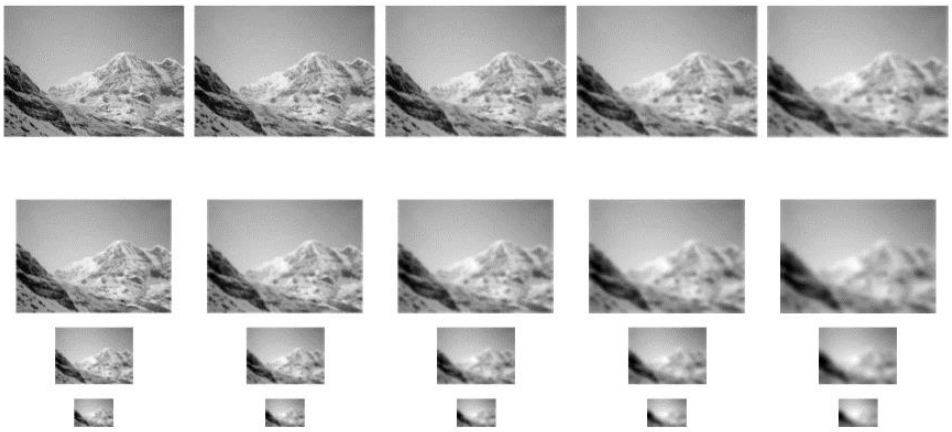

Fig 1. The image pyramid of the template image

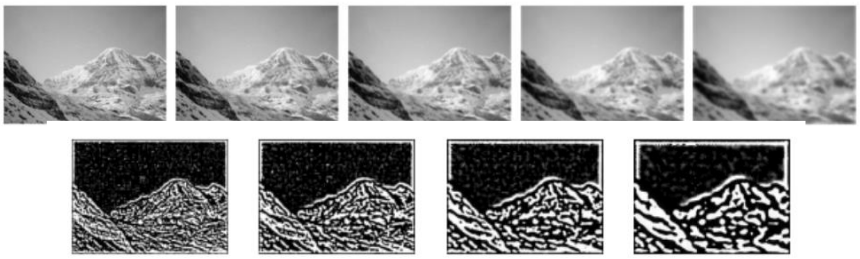

Fig 2. The DoG image at second octave of the template image

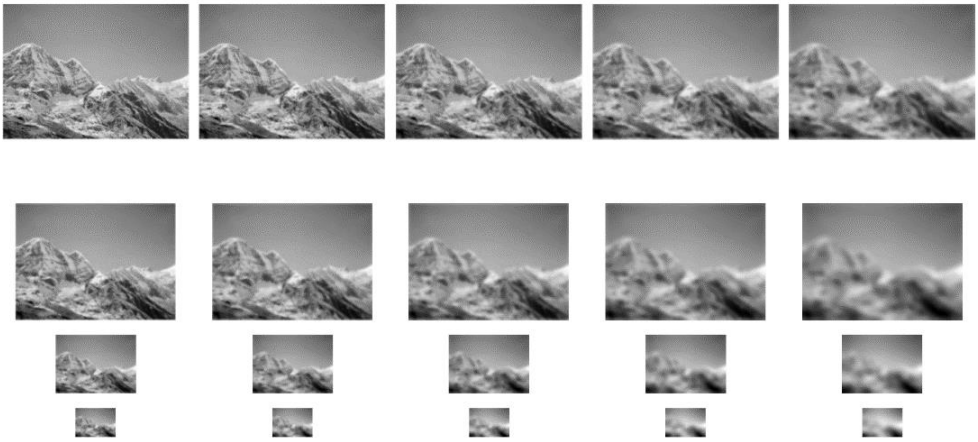

Fig 3. The image pyramid of the scene image

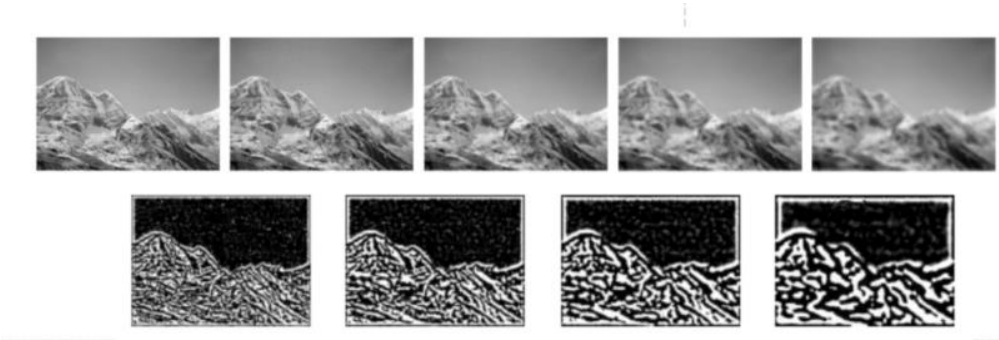

Fig 4. The DoG image at second octave of the scene image

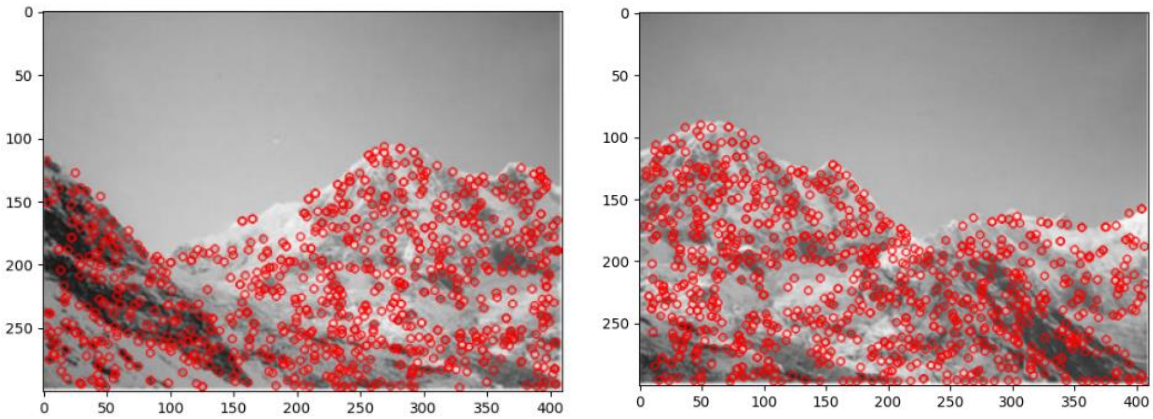

Fig 5. Images with keypoints

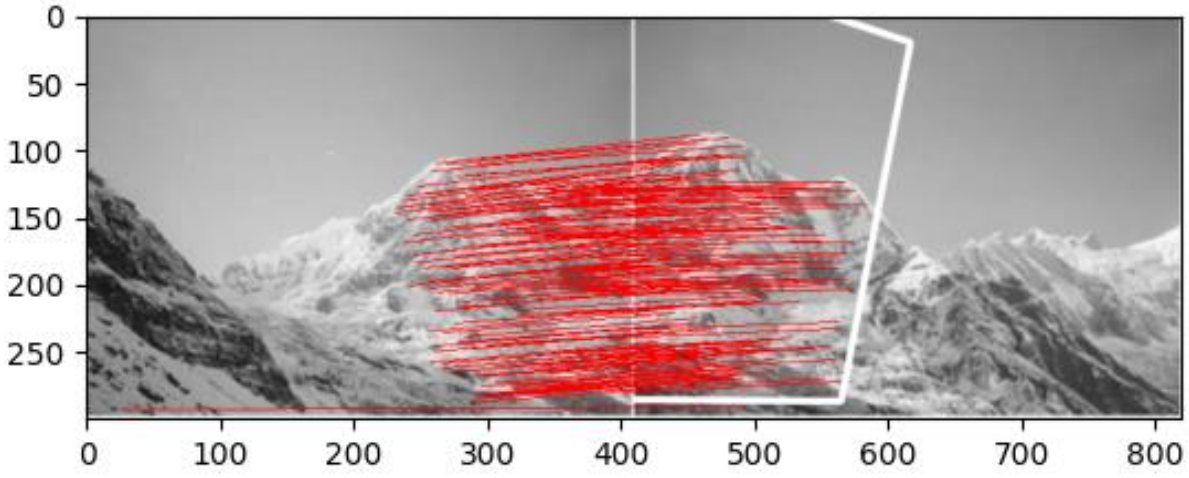

Fig 6. The matching result

b. bomboo_fox Set

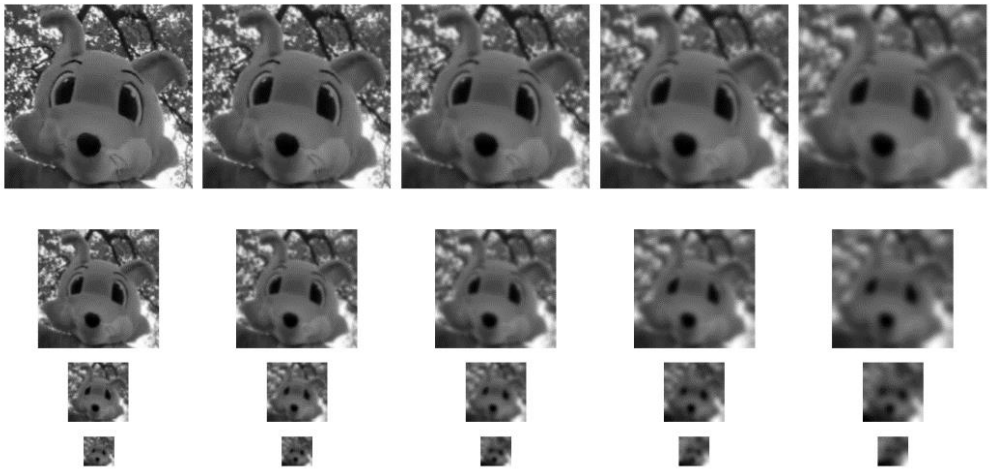

Fig 7. The image pyramid of the template image

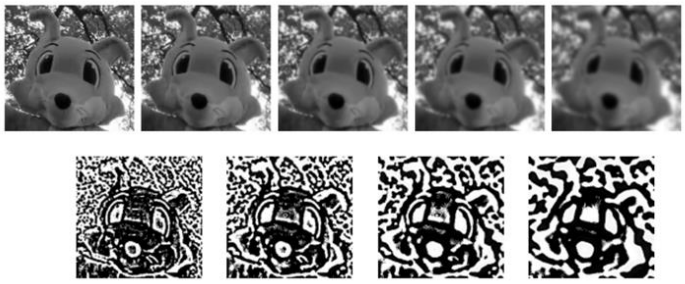

Fig 8. The DoG image at second octave of the template image

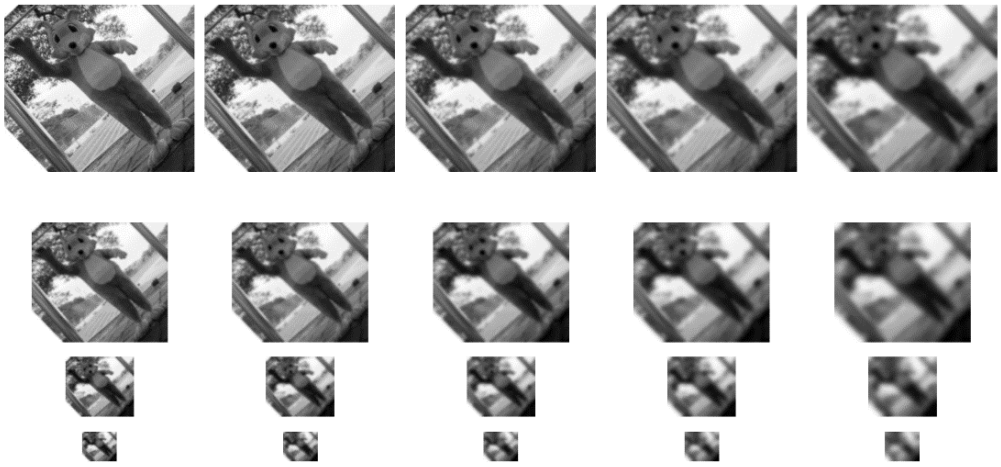

Fig 9. The image pyramid of the scene image

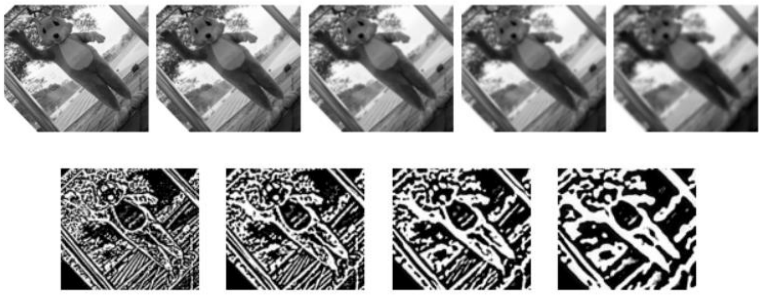

Fig 10. The DoG image at second octave of the scene image

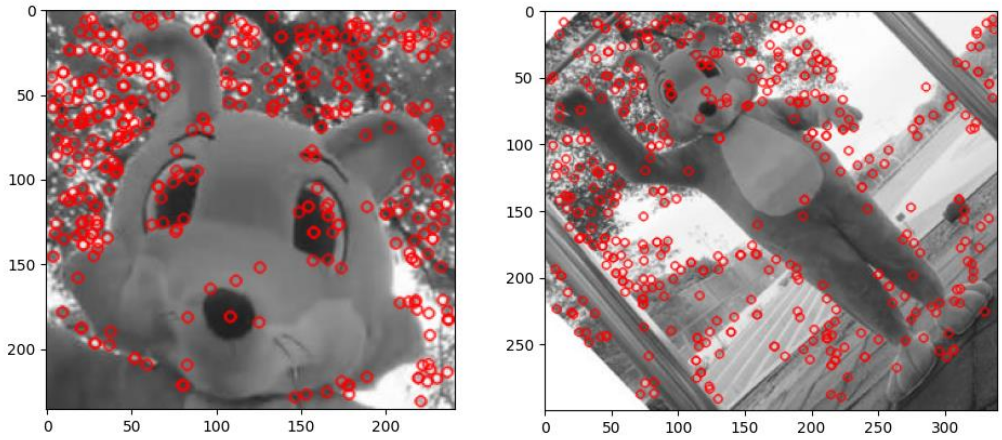

Fig 11. Images with keypoints

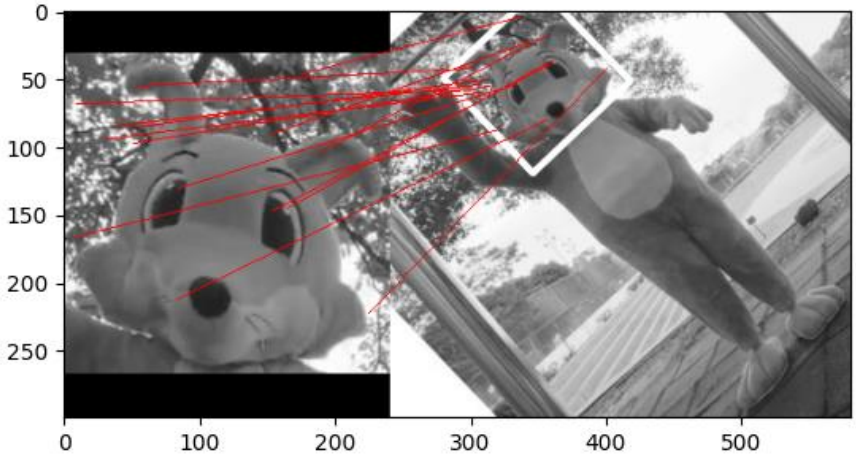

Fig 12. The matching result

c. tree Set

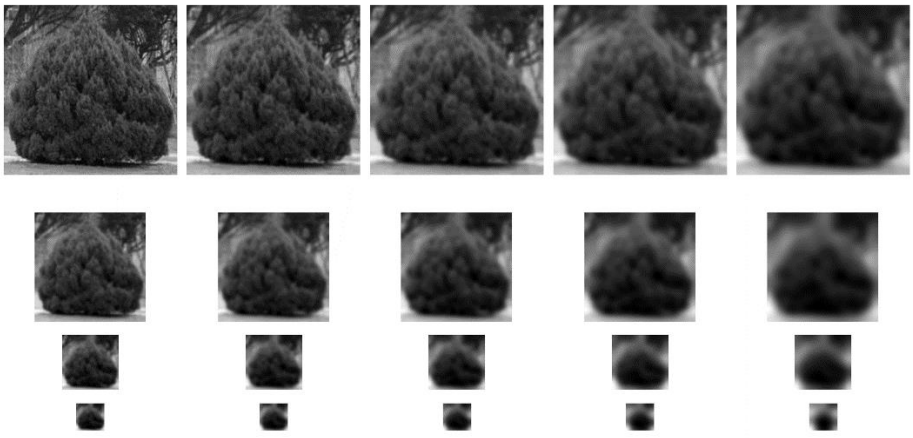

Fig 13. The image pyramid of the template image

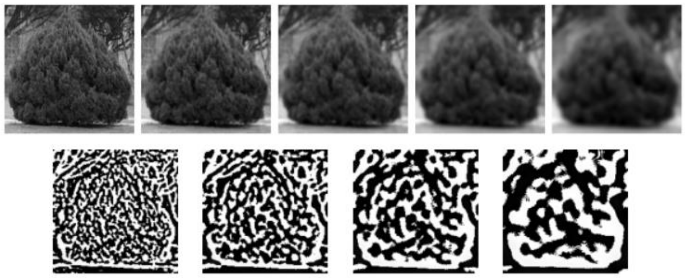

Fig 14. The DoG image at second octave of the template image

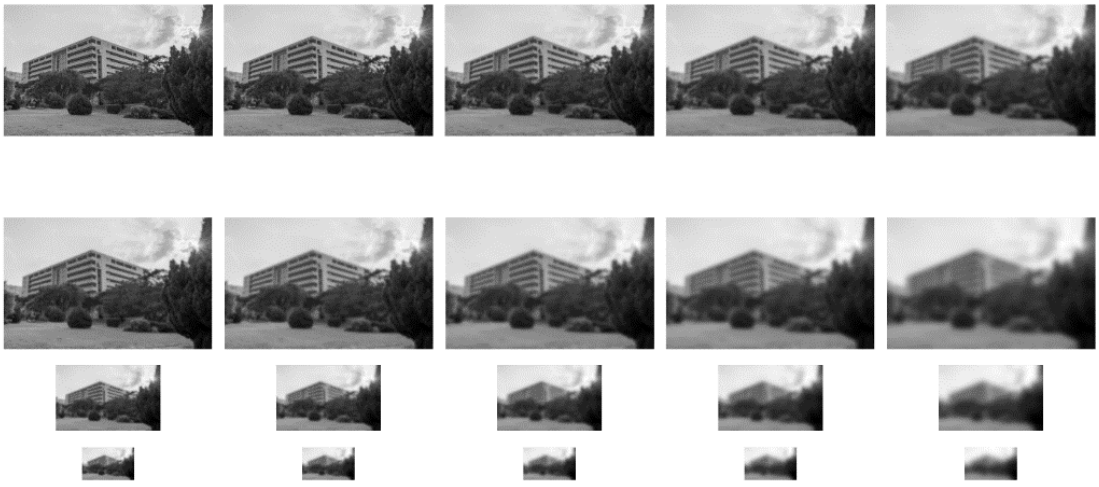

Fig 15. The image pyramid of the scene image

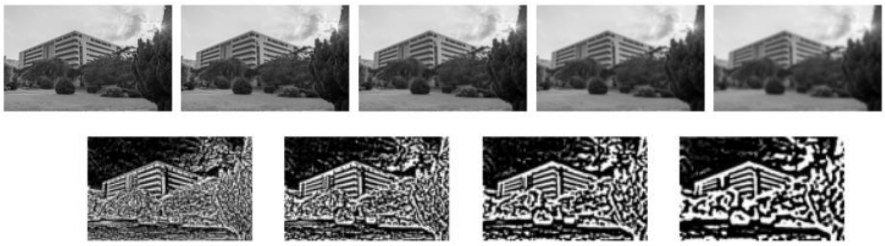

Fig 16. The DoG image at second octave of the scene image

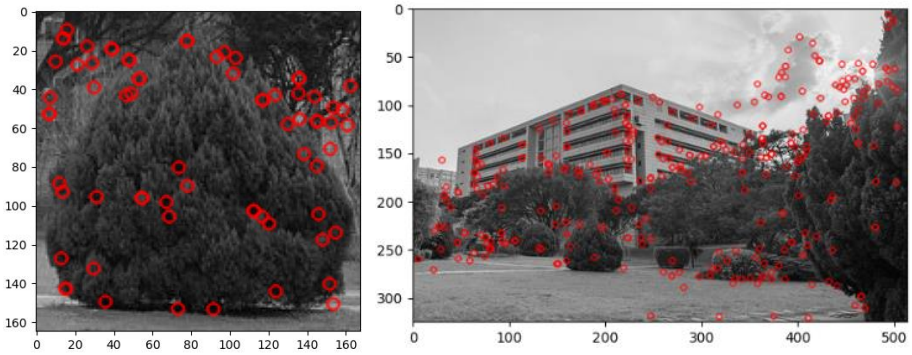

Fig 17. Images with keypoints

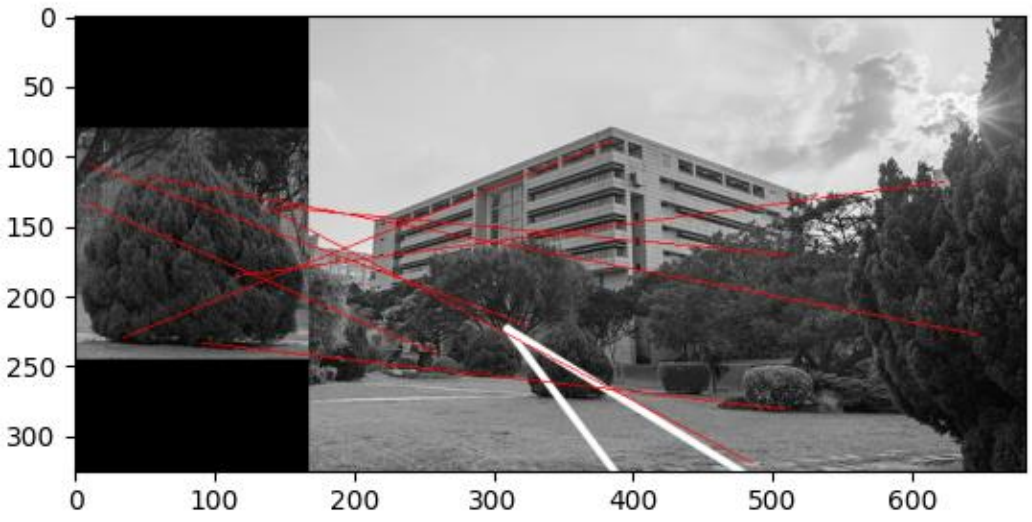

Fig 2. The matching result

- Note: the ratio of closest-distance = 0.7 CAN NOT get the match because no keypoint satisfied with this threshold. This result I use the ratio of closest-distance = 0.9. But the result still NOT good.

d. my test set

Fig 19. The image pyramid of the template image

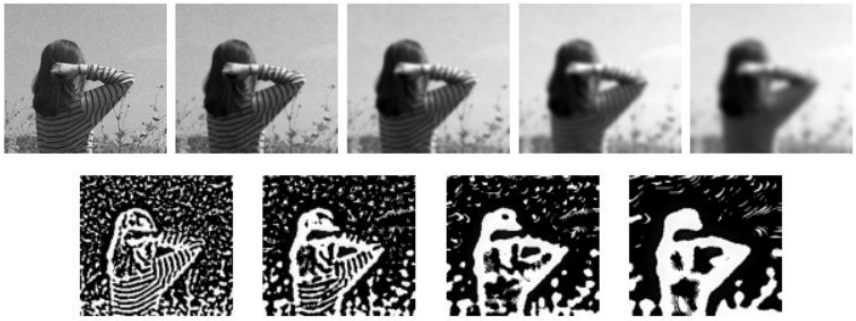

Fig 20. The DoG image at second octave of the template image

Fig 21. The image pyramid of the scene image

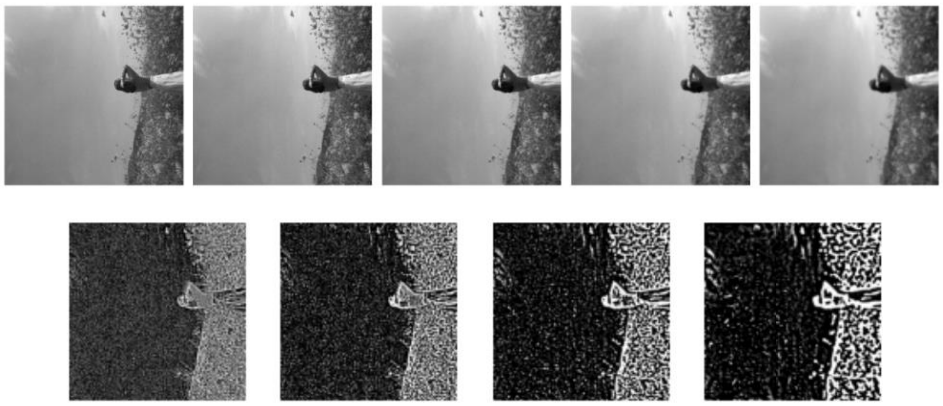

Fig 22. The DoG image at second octave of the scene image

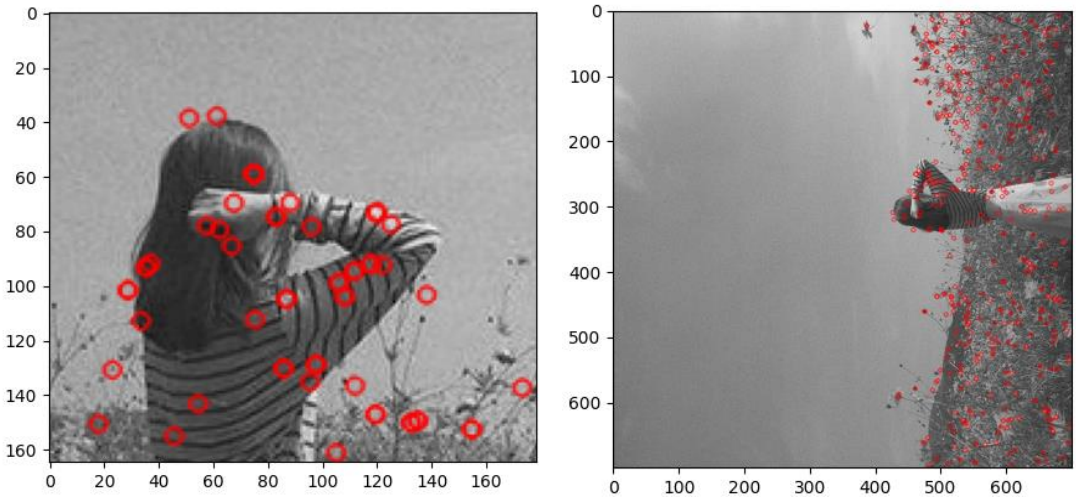

Fig 23. Keypoints images

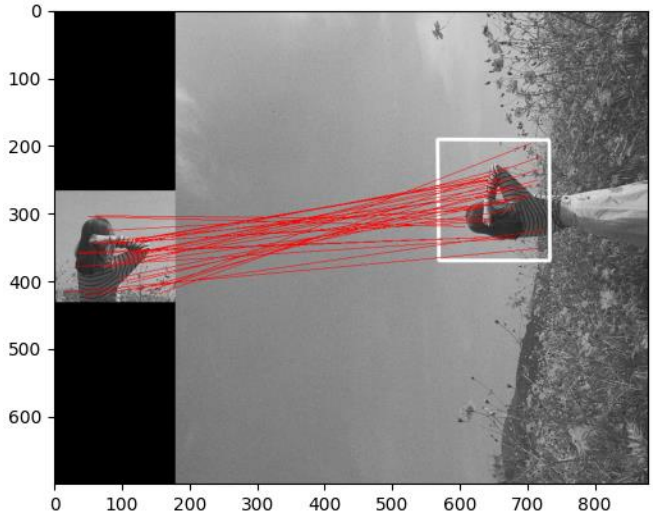

Fig 23. The matching result