Step 1. Initialization:

Write a function: init_images(imgFilename1, imgFilename2) to read in two images, and convert them into grayscale images; return the grayscale images

Step 2. Corner Extraction:

Write a function named

extract_keypts_Harris(img, thresh_Harris=0.005, nms_size=10)

to extract Harris corners from image. Here you can use thresh_Harris=0.005 to threshold the Harris response value (only value > 0.005 will be considered as a corner candidate); nms_size is for non-maximal suppression, we only keep the local maxima within a local 21×21 window (21=2*10+1)

1. Use the built-in function from OpenCV:

   cImg = cv2.cornerHarris(image, blockSize, ksize, k)

   here you can use blockSize=5, ksize=5, and k=0.04;

   The output cImg is a matrix of floating points, recording the response function values.

2. When doing a non-maximal suppression, to avoid problem near the image boundary, we need to do a padding. Write a function named padding(img, padSize), to pad an image (or matrix) with 0 outside its border:
   a. First create a zero matrix using
      img_pad = np.zeros((sizeRow,sizeCol), img.dtype)
      where sizeRow and sizeCol should be the right dimension of the padded image. Note that img_pad has the same element type of img.
   b. Then copy the original image into the middle of the img_pad
   c. Return img_pad
3. Do the non-maximal suppression: for a pixel whose response value > thresh_Harris, it becomes a corner candidate. Check whether it is a local maximal within a local $(2k+1)\times(2k+1)$ window. If so, add its coordinates and response value (x,y,R) into a list called Corners.
4. Finally, extract_keypts_Harris returns the list corners

Step 3. Write a function named matchKeyPts(img1, img2, patchSize, corners1, corners2, maxScoreThresh) to create the correspondence between the two lists of corners (extracted from img1 and img2 respectively).

1. Before doing patch comparison, again we need to pad the image. Call the padding function to extend the image by the size of patchSize.

2. Write a function named score_ZNCC(patch1, patch2) to calculate the similarity score between patch1 and patch2 using zero-mean normalized cross correlation. Before doing a “dot-product”, remember to first subtract its mean element value and normalize it.
3. Use a 2-dimensional for loop to check each pair of corners. For each corner in corners1, find its best corresponding corner in corners2. If this best ZNCC score is bigger than maxScoreThresh, add it into the final correspondence list match.
4. Finally return match.
5. In this homework, you can set patchsize=15, and maxScoreThresh=0.98

Verification Step: Visualize the matching results

I provided a function, called draw_matches(img1, img2, corners1, corners2, matches), for you to generate the matching image.

Simply call draw_matches(img1, img2, corners1, corners2, matches) and provide the images, corner lists, and the match respectively. You will see a ‘cornerMatching.png’ image, which shows the correspondence.