Challenge 6

Student: Xabier Oyanguren Asua
NIU: 1456628 Date: 24/04/2022

1 Objectives

Using Python and some libraries like numpy and scipy (listed in the fifth section), we detected interesting corner-like keypoints in a pair of images and proceeded to automatically find the brute-force matches using a window surrounding each keypoint as descriptor.

2 Data Acquisition

Images were acquired using the camera of a Xiaomi Redmi 5 Plus, and downscaled using local averaging to a third of the original sizes (to a 240x427 shape). The two employed images can be seen in Figure 1.

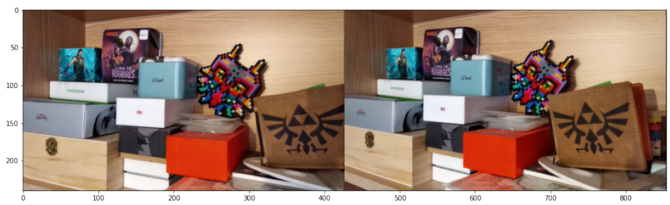

Figure 1: Captured pair of images for the exercise.

3 Procedure

Following the Harris detector principles exposed in the course notes, the Python code attached in the next section was implemented to find corner keypoints of the gray-scale version of the two taken images. The resulting keypoints for each of a set of four different analysed scales can be found in Figure 2. The markers indicating the keypoints have a size proportional to the scale of the Harris detector that found them. The scales are found in the legend of the figure.

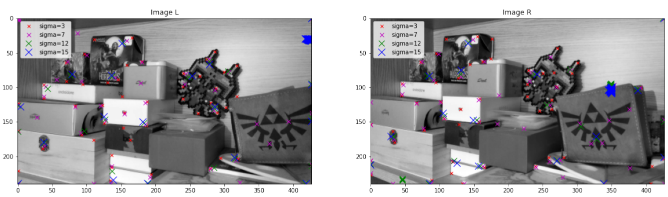

Figure 2: Detected keypoints with the implemented Harris detector for several different pixel scales, which can be found in the legend.

Then, each keypoint is attributed a descriptor of its locality, as a coloured window with a fixed size (the employed parameters can be found in the code section). The pairwise Euclidean distance matrix for all the keypoints in one image relative to the keypoints of the other image are computed. Then, for each keypoint in one image, its matching keypoint of the other image is selected as the one with the smallest descriptor Euclidean distance to it. The resulting pair of associations can be found in Figure 3.

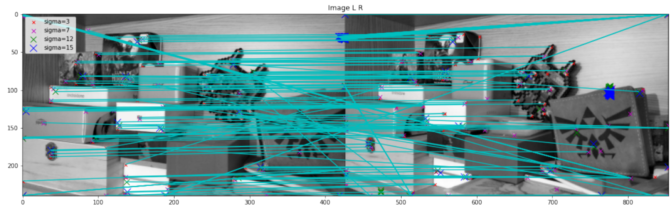

Figure 3: Suggested matches for the keypoints in the two images, using a brute force minimization of the local window descriptor.

In order to filter the results, we then iterate over the matches and only leave those with a matching Euclidean distance for the local window bigger than a certain threshold, and an Euclidean distance in terms of keypoint pixel positions that is smaller than another certain threshold.

4 Results

The resulting keypoint matches can be seen in Figure 4.

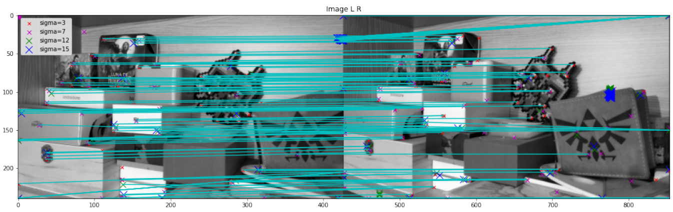

Figure 4: Filtered matches employing the minimized metric for the match and the keypoint index Euclidean distance.

References

[1] Class notes.

5 Packages

Employed packages and external functions were imported as:

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
from scipy.signal import convolve2d
import scipy.ndimage
from skimage.transform import
    downscale_local_mean
from scipy.spatial import distance_matrix
```

6 Code

The functions generated for the Harris detector are:

```
def gaussian1D(sigma, xmin, xmax, N=None, mu=0):
    if N is None:
        N=int(xmax-xmin +1)
    x = np.linspace(xmin, xmax, N)
    g = np.exp(-0.5*x**2/sigma**2)
    return g/g.sum()

def harris_keypoints(im, k=0.05, window_sigmas=[3, 10, 20, 40], threshold_R=1000, best_N_keypoints=100, max_local_wide=100):
    if len(im.shape)==3:
        im = cv2.cvtColor(im, cv2.COLOR_RGB2GRAY)
    dx = np.array([[1,0,-1]])/2.0
    dy = dx.T
    # compute derivatives
    im_dx = convolve2d( im.astype(np.float64), dx, mode='same', boundary='wrap' )
    im_dy = convolve2d( im.astype(np.float64), dy, mode='same', boundary='wrap' )
    # generate metric tensor
    im_metric_tens = np.dstack([im_dx**2, im_dy**2, im_dx*im_dy])
    keypoints={}
    for window_sigma in window_sigmas:
        # smoothing using gaussian window
        g1D = gaussian1D(window_sigma, -window_sigma*3, window_sigma*3)
        im_metric_tens_smooth = np.zeros(im.shape+(3,), dtype=np.float64)
        for j in range(3):
            im_metric_tens_smooth[:,:,j] = convolve2d(
                                convolve2d( im_metric_tens[:,:,j], g1D[:,np.newaxis], mode='same', boundary='symm'),
                                            g1D[np.newaxis, :], mode='same', boundary='symm')
        # compute response function to corners
        dets = im_metric_tens_smooth[:,:,0]*im_metric_tens_smooth[:,:,1]-im_metric_tens_smooth[:,:,2]**2
        traces = im_metric_tens_smooth[:,:,0]+im_metric_tens_smooth[:,:,1]
        R = dets-k*traces**2

        # detect corners thresholding big positive R values
        thresholded = (R>=threshold_R)*R
        local_maxima = scipy.ndimage.filters.maximum_filter( thresholded,
                                                size=(max_local_wide,max_local_wide) ) # size is the size of the locality
        local_max_mask = (thresholded == local_maxima) # convert local max values to binary mask

        # look for the N biggest R keypoints (most clear ones)
        N = min(best_N_keypoints, np.sum(local_max_mask)) # in case we found less
        best_indices = np.unravel_index( np.argsort((local_max_mask*R).flatten())[-N:], shape=local_max_mask.shape )
        keypoints[f'sigma={window_sigma}']=np.asarray(best_indices).T #[N, 2 (h,w)]

    return keypoints
```

For obtaining the descriptor local windows:

```
def get_as_descriptor_windows_around(image, keypoints, window_side):
    assert(window_side%2!=0)
    step = window_side//2
    image = np.pad(image, ((step, step), (step, step), (0,0)), 'edge')
    descriptors = {'keypoints':[], 'flatten_surrounding_window':[], 'scale':[]}
    for key, indices in keypoints.items():
        for indexh, indexw in (indices+step):
            descriptors['keypoints'].append([indexh-step, indexw-step])
            descriptors['scale'].append(float(key.split('=')[-1]))
            descriptors['flatten_surrounding_window'].append(
                (image[(indexh-step):(indexh+step+1), (indexw-step):(indexw+step+1)]).flatten() )
    return descriptors
```

For the brute force matching:

```
def brute_force_matching(descriptorsL, descriptorsR):
    d_mat = distance_matrix(np.array(descriptorsL["flatten_surrounding_window"]),
                    np.array(descriptorsR["flatten_surrounding_window"]), p=2) #[N_R, N_L]
    descriptorsL['matches_idx'] = np.argmin(d_mat, axis=1)
    descriptorsR['matches_idx'] = np.argmin(d_mat, axis=0)
    descriptorsL['matches_keypoints'] = np.array(descriptorsR['keypoints'])[descriptorsL['matches_idx']]
    descriptorsR['matches_keypoints'] = np.array(descriptorsL['keypoints'])[descriptorsR['matches_idx']]
    descriptorsL['match_distance'] = np.min(d_mat, axis=1)
    descriptorsR['match_distance'] = np.min(d_mat, axis=0)
    descriptorsL['keypoint_distance'] = np.sum((np.array(descriptorsL['keypoints'])-descriptorsL['matches_keypoints'])**2, axis=1)
    descriptorsR['keypoint_distance'] = np.sum((np.array(descriptorsR['keypoints'])-descriptorsR['matches_keypoints'])**2, axis=1)

    return descriptorsL, descriptorsR

descriptorsL, descriptorsR = brute_force_matching(descriptorsL, descriptorsR)
```

Then the procedure for the generation of the images was:

```python
imL_col = downscale_local_mean(cv2.imread("imL.jpg")[:,:,::-1], factors=(3,3,1)).astype(np.uint8)
imR_col = downscale_local_mean(cv2.imread("imR.jpg")[:,:,::-1], factors=(3,3,1)).astype(np.uint8)
imL = cv2.cvtColor(imL_col, cv2.COLOR_RGB2GRAY )
imR = cv2.cvtColor(imR_col, cv2.COLOR_RGB2GRAY )
plot(np.hstack((imL_col, imR_col)), size=(30,5), save="one.png")

sigmas = [3, 7, 12, 15]
keypointsL = harris_keypoints(imL, k=0.05, window_sigmas=sigmas, threshold_R=2000,
                              best_N_keypoints=50, max_local_wide=20)
keypointsR = harris_keypoints(imR, k=0.05, window_sigmas=sigmas, threshold_R=2000,
                              best_N_keypoints=50, max_local_wide=20)

sizes=[5, 7, 10, 12]
colors=['r', 'm', 'g', 'b']
fig, axs = plt.subplots(1,2,figsize=(20, 10))
axs[0].imshow(imL, cmap='gray')
axs[1].imshow(imR, cmap='gray')
for j in range(len(sigmas)):
    idxs = keypointsL[f'sigma={sigmas[j]}']
    axs[0].plot(idxs[:,1], idxs[:,0], 'x', markersize=sizes[j], label=f"sigma={sigmas[j]}", color=colors[j])
    idxs = keypointsR[f'sigma={sigmas[j]}']
    axs[1].plot(idxs[:,1], idxs[:,0], 'x', markersize=sizes[j], label=f"sigma={sigmas[j]}", color=colors[j])
axs[0].legend()
axs[0].set_title("Image L")
axs[1].legend()
axs[1].set_title("Image R")
plt.savefig("two.png")
plt.show()

descriptorsL = get_as_descriptor_windows_around(imL_col, keypointsL, window_side=51)
descriptorsR = get_as_descriptor_windows_around(imR_col, keypointsR, window_side=51)


descriptorsL, descriptorsR = brute_force_matching(descriptorsL, descriptorsR)

sizes=[5, 7, 10, 12]
colors=['r', 'm', 'g', 'b']
fig, axs = plt.subplots(1,1,figsize=(20, 10))
axs.imshow(np.hstack((imL,imR)), cmap='gray')
for j in range(len(sigmas)):
    idxs = keypointsL[f'sigma={sigmas[j]}']
    axs.plot(idxs[:,1], idxs[:,0], 'x', markersize=sizes[j], color=colors[j])
    idxs = keypointsR[f'sigma={sigmas[j]}']
    axs.plot( imL.shape[1]+idxs[:,1],idxs[:,0], 'x', markersize=sizes[j], label=f"sigma={sigmas[j]}", color=colors[j])
for keypointL, keypointR in zip(descriptorsL['keypoints'], descriptorsL['matches_keypoints']):
    axs.plot([keypointL[1], imL.shape[1]+keypointR[1]], [keypointL[0], keypointR[0]], color='c')
axs.legend()
axs.set_title("Image L R")
plt.savefig("three.png")
plt.show()

# FILTERING
sizes=[5, 7, 10, 12]
colors=['r', 'm', 'g', 'b']

fig, axs = plt.subplots(1,1,figsize=(20, 10))
axs.imshow(np.hstack((imL,imR)), cmap='gray')
for j in range(len(sigmas)):
    idxs = keypointsL[f'sigma={sigmas[j]}']
    axs.plot(idxs[:,1], idxs[:,0], 'x', markersize=sizes[j], color=colors[j])
    idxs = keypointsR[f'sigma={sigmas[j]}']
    axs.plot( imL.shape[1]+idxs[:,1],idxs[:,0], 'x', markersize=sizes[j], label=f"sigma={sigmas[j]}", color=colors[j])
for keypointL, keypointR, match_metric, keypoint_distance in zip(descriptorsL['keypoints'],
                         descriptorsL['matches_keypoints'], descriptorsL['match_distance'],
                                                     descriptorsL['keypoint_distance']):
    if match_metric<80**2 and keypoint_distance<50**2:
        axs.plot([keypointL[1], imL.shape[1]+keypointR[1]], [keypointL[0], keypointR[0]], color='c')
axs.legend()
axs.set_title("Image L R")
plt.savefig("four.png")
plt.show()
```