SURNAME: Benvenuto NAME: Giulia

I cleared all the outputs because otherwise the size of the notebook was too big to be uploaded on aulaweb.

In [ ]:
```
#from __future__ import division # uncomment this if using Python 2.7
import numpy as np
import math
from scipy import signal, spatial
import matplotlib.pyplot as plt
from matplotlib import cm
from skimage import data, color, img_as_float, img_as_ubyte, filters, feature, util, io
from sklearn import metrics

%matplotlib inline
from IPython.display import HTML, display
```

Feature matching

In this lab we are going to dive deeper into *image matching*, specifically on a local approach based on features correspondances.

We first consider the simplest situation possible: the image pairs we consider are related by simple transformations, therefore we will make the following choices:

1. Feature detector: corners (with the shi-tomasi algorithm)
2. Feature description: patches around the corner (size $w \times w$)
3. Matching strategy: affinity matrix with a similarity measure of choice

The parameters of every intermediate step, must be specified as input arguments. Try with different distance metrics (e.g. euclidean, correlation, squared euclidean) and with all three image pairs (Rubik, Shrub, Phone)

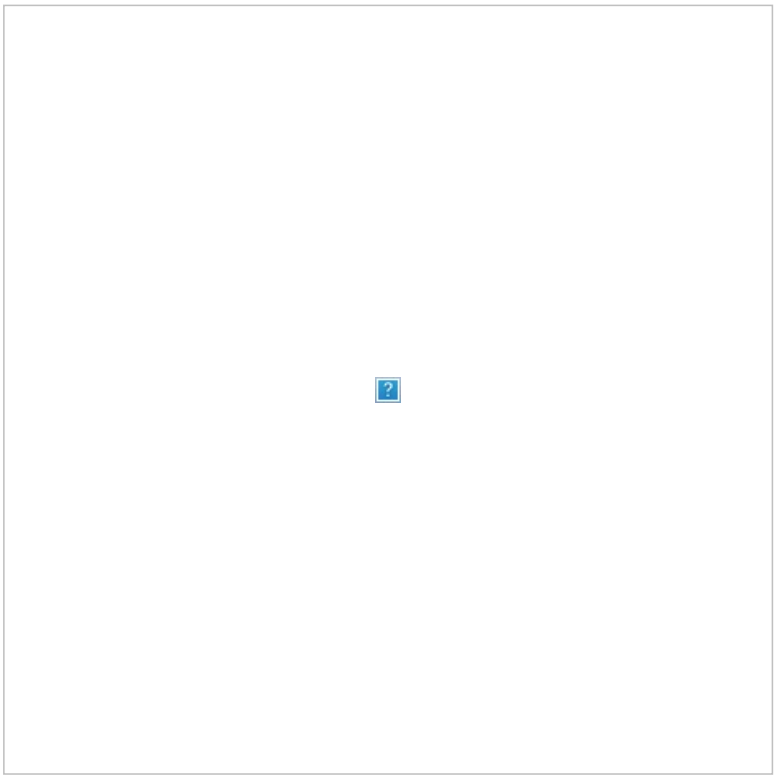
Let us define the main functions

We first write a function that extracts patches of image around each corner (w.r.t. the patch size, that is a parameter)

- First we need to pad our image, adding a surrounding frame of appropriate width. In this way even border features (like the red one in the drawing) will have their neighbourhood

Then, for each corner,

- we adjust the corner coordinates with respect to the new padded image I_ext
- we extract the size_w X size_w patch surrounding the corner (check the range notation)
- we flatten the patch to form a 1D feature vector of size 2*size_w and save it in a list



In [ ]:
```
def patch_descriptor(I, corner_pos, size_w):
    """Extract square patches around each corner on an input grayscale image.
    - I: input RGB image
    - corner_pos: list with position of n corners (row,col)
    - size_w: (integer) patch side
   """

    n = len(corner_pos) # Number of features
    hw = int(np.floor(size_w/2)) # half size of the patch (useful to center the pat)
    I_ext = np.pad(I, hw, 'reflect') # pad the image with a frame of width hw

    # initialize patches list
    patches = np.zeros([n,(2*hw+1)**2])

    for i in range(0, n):

        r = corner_pos[i,0]+hw # adjust the row of each corner considering the padding
        c = corner_pos[i,1]+hw # do the same for the column
        tmp = I_ext[r-hw:r+hw+1, c-hw:c+hw+1] # estract the patch
        patches[i,:] = tmp.flatten() #flatten the patch and save it in the patches list

    return patches
```

Implement the function that computes the feature matching procedure. Assuming we have m features in image 1 and n features in image 2, the affinity matrix will have size $m \times n$. For each feature in image 1, we will be matching the one in image 2 *minimizes* the distance.

Below you find a simple version of the procedure

1. First compute the affinity matrix
2. Then detect the maxima of the affinity matrix
3. Derive che corresponding matches
4. Return the matches and the affinity matrix

Hint: First check `spatial.distance.cdist` from `scipy` module, with Euclidean metric. Then use the formulae below to compute the elements of an affinity matrix (values in the range [0,1] $\exp^{-d(f_i,g_j)/2\sigma^2}$)

In [ ]:
```
def spectral_matching_euclidean(sigma, patches1, patches2, corner_pos1, corner_pos2):

    # evaluate the distance among patches [HERE WE ARE JUST USING THE APPEARANCE SIMILARITY]
    D = spatial.distance.cdist(patches1, patches2, metric='euclidean')
    # compute the affinity matrix using the exponent formulation
    E = np.exp(-D//(2*sigma*sigma))
    # find the minimum distace
    argmaxE_h = np.argmax(E, axis=1) #axis=1 : rows
    argmaxE_v = np.argmax(E, axis=0) #axis=0 : column
    match = []

    # plot the matching pairs (match includes the match from corner_pos2 associated with corner_pos1)
    match = corner_pos2[argmaxE_h,:]

    return E, match
```

Affinity matrix computed by using: 'correlation' as distance metric.

In [ ]:
```
def spectral_matching_correlation(sigma, patches1, patches2, corner_pos1, corner_pos2):

    # evaluate the distance among patches [HERE WE ARE JUST USING THE APPEARANCE SIMILARITY]
    D = spatial.distance.cdist(patches1, patches2, metric='correlation')
    # compute the affinity matrix using the exponent formulation
    E = np.exp(-D//(2*sigma*sigma))
    # find the minimum distace
    argmaxE_h = np.argmax(E, axis=1) #axis=1 : rows
    argmaxE_v = np.argmax(E, axis=0) #axis=0 : column
    match = []

    # plot the matching pairs (match includes the match from corner_pos2 associated with corner_pos1)
    match = corner_pos2[argmaxE_h,:]
    return E, match
```

Affinity matrix computed by using: 'sqeuclidean' as distance metric.

In [ ]:
```
def spectral_matching_sqeuclidean(sigma, patches1, patches2, corner_pos1, corner_pos2):

    # evaluate the distance among patches [HERE WE ARE JUST USING THE APPEARANCE SIMILARITY]
    D = spatial.distance.cdist(patches1, patches2, metric='sqeuclidean')
    # compute the affinity matrix using the exponent formulation
    E = np.exp(-D//(2*sigma*sigma))
    # find the minimum distace
```

```
    argmaxE_h = np.argmax(E, axis=1) #axis=1 : rows
    argmaxE_v = np.argmax(E, axis=0) #axis=0 : column
    match = []

    # plot the matching pairs (match includes the match from corner_pos2 associated with corner_pos1)
    match = corner_pos2[argmaxE_h,:]
    return E, match
```

Write a function that plots the images side-by-side and superimposes the features (**hint:** beware of the offset!) and a line connecting the matched pairs

In [ ]:
```
def show_match(match,corner_pos1,corner_pos2,img1, img2):
    """show match on side-by-side images"""

    img= np.concatenate([img1,img2],axis=1)
    plt.imshow(img, cmap=cm.gist_gray)

    #for pair in match:
    for i in range(0, len(corner_pos1)):
        plt.plot([corner_pos1[i,1], match[i,1]+img1.shape[1]], [corner_pos1[i,0], match[i,0]],'y')
        #plt.plot([corner_pos1[pair[0],1], corner_pos2[pair[1],1]+img1.shape[1]], [corner_pos1[pair[0],0], corn


    plt.scatter(corner_pos1[:,1], corner_pos1[:,0], s=10, c='r')
    plt.scatter(corner_pos2[:,1]+img1.shape[1], corner_pos2[:,0], s=20, c='b')

    return img
```

Let us test the feature matching pipeline

We start with a simple image pair. We first load them and visualize them

In [ ]:
```
# LOAD IMAGES
RGBimg1 = io.imread('images/shrub_L.jpg')
img1 = img_as_float(color.rgb2gray(RGBimg1))

RGBimg2 = io.imread('images/shrub_R.jpg') #(See below)
img2 = img_as_float(color.rgb2gray(RGBimg2))

#PLOT THEM
plt.figure(figsize=(12,6))
plt.subplot(121)
plt.imshow(RGBimg1, cmap=
           cm.gist_gray)
plt.title('Image 1')
plt.subplot(122)
plt.imshow(RGBimg2, cmap=cm.gist_gray)
plt.title('Image 2');
```

Now, let us identify the corners by using the shi tomasi algorithm implemented in skimage

In [ ]:
```
# FEATURE DETECTION
# using the shi-tomasi algorithm, identify the corners in both images

corners1 = feature.corner_peaks(feature.corner_shi_tomasi(img1), num_peaks = 100) # IT MAY BE WORTH ADDING num_
corners2 = feature.corner_peaks(feature.corner_shi_tomasi(img2), num_peaks = 100) # IT MAY BE WORTH ADDING num_

# plot the results on both images side by side
plt.figure(figsize=(12,6))
plt.subplot(121)
plt.imshow(RGBimg1, cmap=cm.gist_gray)
plt.scatter(corners1[:,1], corners1[:,0], s=30)
plt.title('skimage.feature.corner_peaks result')

plt.subplot(122)
plt.imshow(RGBimg2, cmap=cm.gist_gray)
plt.scatter(corners2[:,1], corners2[:,0], s=30, c='r')
plt.title('skimage.feature.corner_peaks result');
```

In [ ]:
```
# FEATURE DESCRIPTORS
# TRY WITH DIFFERENT WINDOW SIZES
patches1 = patch_descriptor(img1, corners1, 40)
patches2 = patch_descriptor(img2, corners2, 40)
```

Distance metrics: "euclidean"

In [ ]:
```
# FEATURE MATCHING
plt.figure(figsize=(12,6))
```

```
D,match = spectral_matching_euclidean(0.5, patches1, patches2, corners1, corners2)
match_euclidean = show_match(match,corners1,corners2,img1, img2)
```

In [ ]:
```
# LET US ALSO HAVE A LOOK AT THE AFFINITY MATRIX
plt.imshow(D)
plt.colorbar()
```

Observations: here I used the distance metrics "euclidean", this means that the Affinity matrix *D* is computed by taking the euclidean distance between *patch1* and *patch2*. Visualizing the affinity matrix as an image we notice that most of the values are equal to zero or really close to it, this means that most of the features are matched in the right way.

Distance metrics: "correlation"

In [ ]:
```
# FEATURE MATCHING
plt.figure(figsize=(12,6))
D,match = spectral_matching_correlation(0.5, patches1, patches2, corners1, corners2)
match_euclidean = show_match(match,corners1,corners2,img1, img2)
```

In [ ]:
```
# LET US ALSO HAVE A LOOK AT THE AFFINITY MATRIX
plt.imshow(D)
plt.colorbar()
```

Observations: here I used the distance metrics "correlation". The correlation distance metric measures the similarity between two variables by calculating the distance between their normalized values. In this case, differently from what happend on the previous case we have that by visualizing the affinity matrix as an image it is possible to notice that a lot more value are different from zero, which correspond to a bigger number of features matched in the wrong way.

Distance metrics: "sqeuclidean"

In [ ]:
```
# FEATURE MATCHING
plt.figure(figsize=(12,6))
D,match = spectral_matching_sqeuclidean(0.5, patches1, patches2, corners1, corners2)
match_euclidean = show_match(match,corners1,corners2,img1, img2)
```

In [ ]:
```
# LET US ALSO HAVE A LOOK AT THE AFFINITY MATRIX
plt.imshow(D)
plt.colorbar()
```

Try with the other images

In [ ]:
```
# LOAD IMAGES
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

RGBimg1 = io.imread('images/Rubik1.pgm')
img1 = img_as_float(color.rgb2gray(RGBimg1))

RGBimg2 = io.imread('images/Rubik2.pgm') #(See below)
img2 = img_as_float(color.rgb2gray(RGBimg2))

#PLOT THEM
plt.figure(figsize=(12,6))
plt.subplot(121)
plt.imshow(RGBimg1, cmap=cm.gist_gray)
plt.title('Image 1')
plt.subplot(122)
plt.imshow(RGBimg2, cmap=cm.gist_gray)
plt.title('Image 2');
```

In [ ]:
```
# FEATURE DETECTION
# using the shi-tomasi algorithm, identify the corners in both images

corners1 = feature.corner_peaks(feature.corner_shi_tomasi(img1), threshold_rel=0.1)
corners2 = feature.corner_peaks(feature.corner_shi_tomasi(img2), threshold_rel=0.1)

# plot the results on both images side by side
plt.figure(figsize=(12,6))
plt.subplot(121)
plt.imshow(RGBimg1, cmap=cm.gist_gray)
plt.scatter(corners1[:,1], corners1[:,0], s=30)
plt.title('skimage.feature.corner_peaks result')

plt.subplot(122)
plt.imshow(RGBimg2, cmap=cm.gist_gray)
plt.scatter(corners2[:,1], corners2[:,0], s=30, c='r')
plt.title('skimage.feature.corner_peaks result');
```

Window size = 10

- In this case I tried the window size = 10.

This means that the patch side is small and this has as consequence more errors because it's more difficult to match patches in which there are less informations.

In [ ]:
```
# FEATURE DESCRIPTORS
# TRY WITH DIFFERENT WINDOW SIZES
patches1 = patch_descriptor(img1, corners1, 10)
patches2 = patch_descriptor(img2, corners2, 10)
```

In [ ]:
```
# FEATURE MATCHING
plt.figure(figsize=(12,6))
D,match = spectral_matching_euclidean(0.8, patches1, patches2, corners1, corners2)
match_euclidean = show_match(match,corners1,corners2,img1, img2)
```

In [ ]:
```
# LET US ALSO HAVE A LOOK AT THE AFFINITY MATRIX
plt.imshow(D)
plt.colorbar()
```

Window size = 50

- In this case I tried the window size = 50.

This means that the patch side is bigger respect to the case before and this has as consequence less errors because it's less difficult to match patches in which there are more informations that help to distinguish the portion of the image that is sourrounded.

In [ ]:
```
# FEATURE DESCRIPTORS
# TRY WITH DIFFERENT WINDOW SIZES
patches1 = patch_descriptor(img1, corners1, 50)
patches2 = patch_descriptor(img2, corners2, 50)
```

In [ ]:
```
# FEATURE MATCHING
plt.figure(figsize=(12,6))
D,match = spectral_matching_euclidean(0.8, patches1, patches2, corners1, corners2)
match_euclidean = show_match(match,corners1,corners2,img1, img2)
```

In [ ]:
```
# LET US ALSO HAVE A LOOK AT THE AFFINITY MATRIX
plt.imshow(D)
plt.colorbar()
```

Observations:

But we need to note that the *spectral_matching* implemented like above is wrong because it compute the maxima only on the rows and not on both rows and columns so below we need to correct it.

Improvements

There are a number of improvement you could consider at some point. Below you find a mandatory improvement and some optional ones

1. The function spectral_matching does not compute the maxima in the correct way; *correct the function* so that the identified feature pairs are maxima of both rows and colums .

Optionally you may also have a look at the following

1. spectral_matching: you may add a threshold on the distance, setting to 0 the affinity matrix entries below a threshold(see theory)
2. In the affinity matrix computation, one could also include an evaluation on the position of

features (close features should be favoured if we have a prior on image similarity). Again, see theory 2. Corners on the border of the image should be discarded as they tend to be less reliable.

Spectral matching corrected

Now the *spectral_matching2* function check maxima of boths rows and columns.

In [ ]:
```
def spectral_matching2(sigma, patches1, patches2,corner_pos1,corner_pos2):

    # evaluate the distance among patches
    D = spatial.distance.cdist(patches1,patches2, metric='euclidean')
    # compute the affinity matrix using the exponent formulation
    E = np.exp(-D/(2*sigma*sigma))
    # find the minimum distace
```

```
    argmaxR = np.argmax(E, axis=1)
    argmaxC = np.argmax(E, axis=0)

    match = []
    for row, maxC in enumerate(argmaxR):
        if argmaxC[maxC] == row:
            match.append([row, corner_pos2[maxC, 0], corner_pos2[maxC, 1]])
    # plot the matching pairs (match includes the match from corner_pos2 associated with corner_pos1)
    #match = corner_pos2[argmaxE,:]

    return E, np.array(match)
```

In [ ]:
```
def show_match2(match, corner_pos1, corner_pos2, img1, img2):

    # match now has information on also which corner1 the match refer to, because some may not have a match anyr

    img = np.concatenate([img1, img2], axis=1)
    plt.imshow(img, cmap=cm.gist_gray)

    for i in range(match.shape[0]):
        plt.plot([corner_pos1[ match[i, 0], 1 ], match[i, 2] + img1.shape[1]], [corner_pos1[ match[i, 0], 0], ma

    plt.scatter(corner_pos1[:,1], corner_pos1[:,0], s=10)
    plt.scatter(corner_pos2[:,1]+img1.shape[1], corner_pos2[:,0], s=20, c='r')

    return img
```

In [ ]:
```
# LOAD IMAGES
RGBimg1 = io.imread('images/shrub_L.jpg')
img1 = img_as_float(color.rgb2gray(RGBimg1))

RGBimg2 = io.imread('images/shrub_R.jpg')
img2 = img_as_float(color.rgb2gray(RGBimg2))

#PLOT THEM
plt.figure(figsize=(12,6))
plt.subplot(121)
plt.imshow(RGBimg1, cmap=cm.gist_gray)
plt.title('Image 1')
plt.subplot(122)
plt.imshow(RGBimg2, cmap=cm.gist_gray)
plt.title('Image 2');
```

In [ ]:
```
corners1 = feature.corner_peaks(feature.corner_shi_tomasi(img1), threshold_rel=0.08)
corners2 = feature.corner_peaks(feature.corner_shi_tomasi(img2), threshold_rel=0.08)

# plot the results on both images side by side
plt.figure(figsize=(12,6))
plt.subplot(121)
plt.imshow(RGBimg1, cmap=cm.gist_gray)
plt.scatter(corners1[:,1], corners1[:,0], s=30)
plt.title('skimage.feature.corner_peaks result')

plt.subplot(122)
plt.imshow(RGBimg2, cmap=cm.gist_gray)
plt.scatter(corners2[:,1], corners2[:,0], s=30, c='r')
plt.title('skimage.feature.corner_peaks result');
```

In [ ]:
```
patches1 = patch_descriptor(img1, corners1, 40)
patches2 = patch_descriptor(img2, corners2, 40)
```

In [ ]:
```
# FEATURE MATCHING
plt.figure(figsize=(12,6))
D,match = spectral_matching2(0.5, patches1, patches2, corners1, corners2)
match_euclidean = show_match2(match,corners1,corners2,img1, img2)
```

In [ ]:
```
# LET US ALSO HAVE A LOOK AT THE AFFINITY MATRIX
plt.imshow(D)
plt.colorbar()
```

Observations: with the corrected function *spectral_matching2* the result is better. Now the feature matching is way more precise. In fact if we consider the image with the road sign, respect to the first case with the wrong *spectral_matching*, now the result is much better.

In [ ]:
```
# LOAD IMAGES
RGBimg1 = io.imread('images/phone1.png')
img1 = img_as_float(color.rgb2gray(RGBimg1))

RGBimg2 = io.imread('images/phone2.png')
img2 = img_as_float(color.rgb2gray(RGBimg2))

#PLOT THEM
```

```
plt.figure(figsize=(12,6))
plt.subplot(121)
plt.imshow(RGBimg1, cmap=cm.gist_gray)
plt.title('Image 1')
plt.subplot(122)
plt.imshow(RGBimg2, cmap=cm.gist_gray)
plt.title('Image 2');
```

In [ ]:
```
corners1 = feature.corner_peaks(feature.corner_shi_tomasi(img1), threshold_rel=0.1)
corners2 = feature.corner_peaks(feature.corner_shi_tomasi(img2), threshold_rel=0.1)

# plot the results on both images side by side
plt.figure(figsize=(12,6))
plt.subplot(121)
plt.imshow(RGBimg1, cmap=cm.gist_gray)
plt.scatter(corners1[:,1], corners1[:,0], s=30)
plt.title('skimage.feature.corner_peaks result')

plt.subplot(122)
plt.imshow(RGBimg2, cmap=cm.gist_gray)
plt.scatter(corners2[:,1], corners2[:,0], s=30, c='r')
plt.title('skimage.feature.corner_peaks result');
```

In [ ]:
```
patches1 = patch_descriptor(img1, corners1, 40)
patches2 = patch_descriptor(img2, corners2, 40)
```

In [ ]:
```
# FEATURE MATCHING
plt.figure(figsize=(12,6))
D,match = spectral_matching2(0.5, patches1, patches2, corners1, corners2)
match_euclidean = show_match2(match,corners1,corners2,img1, img2)
```

In [ ]:
```
# LET US ALSO HAVE A LOOK AT THE AFFINITY MATRIX
plt.imshow(D)
plt.colorbar()
```

Experiments

Here are some experiments you should carry out. Add code snippets and comments below. Conclude with a final discussion section.

1. Debug n. 0 is to use the same image for img1 and img2. Answer before proceeding: what do you expect to find? Check the Affinity Matrix
2. How to proceed if you want to obtain fewer corners?
3. Now try out with two different images of the same object, again analyse the matches as well as the affinity matrix. Try out different sigma values: are the results in line with your expectations?
4. now "break" the simple matching procedure by applying appropriate image transformations: use, as inputs, I1 and transformed(I1), where transformed may be:
    - small or large translations
    - small or large rotations
    - small or large zoom in
5. To be sure you fully grasp the concepts, you should try out other image pairs (see the Images folder)

1) Analysis

If we use the same image for img1 and img2 all the corners are matched perfectly because it's like if they're matching with "themselves". I expect the affinity matrix square and diagonal with all the values equal to 1.

In [ ]:
```
# LOAD IMAGES
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

RGBimg1 = io.imread('images/Rubik1.pgm')
img1 = img_as_float(color.rgb2gray(RGBimg1))

RGBimg2 = io.imread('images/Rubik1.pgm')
img2 = img_as_float(color.rgb2gray(RGBimg2))

# FEATURE DETECTION
corners1 = feature.corner_peaks(feature.corner_shi_tomasi(img1), threshold_rel=0.1)
corners2 = feature.corner_peaks(feature.corner_shi_tomasi(img2), threshold_rel=0.1)

# FEATURE DESCRIPTION
patches1 = patch_descriptor(img1, corners1, 40)
```

```
patches2 = patch_descriptor(img2, corners2, 40)

# PLOT THE RESULT
plt.figure(figsize=(12,6))
D,match = spectral_matching2(0.5, patches1, patches2, corners1, corners2)
show_match2(match, corners1, corners2, img1, img2)

# AFFINITY MATRIX and IDENTITY MATRIX
fig, axs = plt.subplots(ncols=2, figsize=(12, 6))
axs[0].imshow(D)
axs[0].set_title("Affinity Matrix")
axs[1].imshow(np.eye(D.shape[0]))
axs[1].set_title("Identity Matrix");
```

In [ ]:
```
print("Diagonal of the Affinity Matrix")
print(D.diagonal())
```

2) Analysis

If we want less corners we might increase the parameter sigma in the shi-tomasi algorithm, or increase the threshold parameter in the function corner_peaks.

sigma = 4

In [ ]:
```
# LOAD IMAGES
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

RGBimg1 = io.imread('images/phone1.png')
img1 = img_as_float(color.rgb2gray(RGBimg1))

RGBimg2 = io.imread('images/phone2.png')
img2 = img_as_float(color.rgb2gray(RGBimg2))

# FEATURE DETECTION
corners1 = feature.corner_peaks(feature.corner_shi_tomasi(img1, sigma = 4), threshold_rel=0.1)
corners2 = feature.corner_peaks(feature.corner_shi_tomasi(img2, sigma = 4), threshold_rel=0.1)

# FEATURE DESCRIPTION
patches1 = patch_descriptor(img1, corners1, 40)
patches2 = patch_descriptor(img2, corners2, 40)

# PLOT THE RESULT
plt.figure(figsize=(12,6))
D, match = spectral_matching2(0.5, patches1, patches2, corners1, corners2)

plt.figure(figsize=(12,6))
plt.subplot(121)
plt.imshow(img1, cmap=cm.gist_gray)
plt.scatter(corners1[:,1], corners1[:,0], s=30)
plt.title('skimage.feature.corner_peaks result')

plt.subplot(122)
plt.imshow(img2, cmap=cm.gist_gray)
plt.scatter(corners2[:,1], corners2[:,0], s=30, c='r')
plt.title('skimage.feature.corner_peaks result');

plt.figure(figsize=(12, 6))
match_euclidean = show_match2(match,corners1,corners2,img1, img2)
plt.subplots()
plt.imshow(D)
plt.colorbar();
```

In [ ]:
```
print("Corners 1:" + str(corners1.shape))
print("Corners 2:" + str(corners2.shape))
```

3) Analysis

The affinity matrix is not square anymore and its entry are very small.

From the value assigned to the "sigma" parameter depend the number of corners detected in fact, in general we have that:

- when you increase the sigma value, the Gaussian filter used for smoothing the image will have a larger standard deviation, leading to a more blurred image. This can result in a reduction in the number of detected corners as well as a decrease in their sharpness.
- when you decrease the sigma value, the Gaussian filter will have a smaller standard deviation, resulting in a sharper image with more detailed corners.

sigma = 1

In [ ]:
```
# LOAD IMAGES
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

RGBimg1 = io.imread('images/phone1.png')
img1 = img_as_float(color.rgb2gray(RGBimg1))

RGBimg2 = io.imread('images/phone2.png')
img2 = img_as_float(color.rgb2gray(RGBimg2))

# FEATURE DETECTION
corners1 = feature.corner_peaks(feature.corner_shi_tomasi(img1, sigma = 1), threshold_rel=0.1)
corners2 = feature.corner_peaks(feature.corner_shi_tomasi(img2, sigma = 1), threshold_rel=0.1)

# FEATURE DESCRIPTION
patches1 = patch_descriptor(img1, corners1, 40)
patches2 = patch_descriptor(img2, corners2, 40)

# PLOT THE RESULT
plt.figure(figsize=(12,6))
D, match = spectral_matching2(0.5, patches1, patches2, corners1, corners2)

plt.figure(figsize=(12,6))
plt.subplot(121)
plt.imshow(img1, cmap=cm.gist_gray)
plt.scatter(corners1[:,1], corners1[:,0], s=30)
plt.title('skimage.feature.corner_peaks result')

plt.subplot(122)
plt.imshow(img2, cmap=cm.gist_gray)
plt.scatter(corners2[:,1], corners2[:,0], s=30, c='r')
plt.title('skimage.feature.corner_peaks result');

plt.figure(figsize=(12, 6))
match_euclidean = show_match2(match,corners1,corners2,img1, img2)
plt.subplots()
plt.imshow(D)
plt.colorbar();
```

In [ ]:
```
print("Corners 1:" + str(corners1.shape))
print("Corners 2:" + str(corners2.shape))
```

Observations: as I expected, using sigma = 1 corresdpond to find more corners respect to the sigma = 4 case.

4) Translation

We can see that the algorithm we applied is resistant to translation. This is expected because the patch descriptor we used don't change with translation.

In [ ]:
```
# LOAD IMAGE
RGBimg1 = io.imread('images/phone1.png')
img1 = img_as_float(color.rgb2gray(RGBimg1))

# TRANSLATION OF THE IMAGE
import skimage.transform as tf
small = tf.warp(img1, tf.SimilarityTransform(translation=(30, 0)))
big = tf.warp(img1, tf.SimilarityTransform(translation=(180, 100)))

# FEATURE DETECTION
corners1 = feature.corner_peaks(feature.corner_shi_tomasi(img1), threshold_rel=0.1)
cornersSmall = feature.corner_peaks(feature.corner_shi_tomasi(small), threshold_rel=0.1)
cornersBig = feature.corner_peaks(feature.corner_shi_tomasi(big), threshold_rel=0.1)
#print(corners1.shape, corners2.shape)

# FEATURE DESCRIPTION
patches1 = patch_descriptor(img1, corners1, 40)
patchesSmall = patch_descriptor(small, cornersSmall, 40)
patchesBig = patch_descriptor(big, cornersBig, 40)

# PLOT THE RESULT
D, match = spectral_matching2(0.5, patches1, patchesSmall, corners1, cornersSmall)
plt.figure(figsize=(12, 6))
match_euclidean = show_match2(match,corners1,cornersSmall,img1, small)


D, match = spectral_matching2(0.5,patches1, patchesBig, corners1, cornersBig)
plt.figure(figsize=(12, 6))
match_euclidean = show_match2(match,corners1,cornersBig,img1, big)
```

4) Rotation

- For a small rotation, the matching are still good.
- For a very significant rotation (90°), very few corners have a match, and those which have are wrong.
- For an even more big rotation (180°), we see that some corners on the numpad of the phone have the correct match, but the ones on the cup don't. This may be due to the symmetrical nature of those corners, which the one on the cup don't have.

In [ ]:
```
# LOAD IMAGE
RGBimg1 = io.imread('images/phone1.png')
img1 = img_as_float(color.rgb2gray(RGBimg1))

# ROTATION OF THE IMAGE
import skimage.transform as tf
small = tf.warp(img1, tf.SimilarityTransform(rotation = np.pi / 15))
big = tf.warp(img1, tf.SimilarityTransform(rotation = np.pi/2, translation=(img1.shape[0], 0 )))
veryBig = tf.warp(img1, tf.SimilarityTransform(rotation = np.pi, translation=(img1.shape[1], img1.shape[0] )))

# FEATURE DETECTION
corners1 = feature.corner_peaks(feature.corner_shi_tomasi(img1), threshold_rel=0.1)
cornersSmall = feature.corner_peaks(feature.corner_shi_tomasi(small), threshold_rel=0.1)
cornersBig = feature.corner_peaks(feature.corner_shi_tomasi(big), threshold_rel=0.1)
cornersVBig = feature.corner_peaks(feature.corner_shi_tomasi(veryBig), threshold_rel=0.1)
#print(corners1.shape, corners2.shape)

# FEATURE DESCRIPTION
patches1 = patch_descriptor(img1, corners1, 40)
patchesSmall = patch_descriptor(small, cornersSmall, 40)
patchesBig = patch_descriptor(big, cornersBig, 40)
patchesVBig = patch_descriptor(veryBig, cornersVBig, 40)

# PLOT THE RESULT
D,match = spectral_matching2(0.5,patches1, patchesSmall, corners1, cornersSmall)
plt.figure(figsize=(12, 6))
match_euclidean = show_match2(match,corners1,cornersSmall,img1, small)

D,match = spectral_matching2(0.5,patches1, patchesBig, corners1, cornersBig)
plt.figure(figsize=(12, 6))
match_euclidean = show_match2(match,corners1,cornersBig,img1, big)

D,match = spectral_matching2(0.5,patches1, patchesVBig, corners1, cornersVBig)
plt.figure(figsize=(12, 6))
match_euclidean = show_match2(match,corners1,cornersVBig,img1, veryBig)
```

4) Zoom in

For small scale variation we don't have problems, but for a large change we see that most of the corners don't have a match.

In [ ]:
```
# LOAD IMAGE
RGBimg1 = io.imread('images/shrub_L.jpg')
img1 = img_as_float(color.rgb2gray(RGBimg1))

# ZOOM IN OF THE IMAGE
small = tf.warp(img1, tf.SimilarityTransform(scale = 0.9))
big = tf.warp(img1, tf.SimilarityTransform(scale = 4))

# FEATURE DETECTION
corners1 = feature.corner_peaks(feature.corner_shi_tomasi(img1), threshold_rel=0.1)
cornersSmall = feature.corner_peaks(feature.corner_shi_tomasi(small), threshold_rel=0.1)
cornersBig = feature.corner_peaks(feature.corner_shi_tomasi(big), threshold_rel=0.1)
#print(corners1.shape, corners2.shape)

# FEATURE DESCRIPTION
patches1 = patch_descriptor(img1, corners1, 40)
patchesSmall = patch_descriptor(small, cornersSmall, 40)
patchesBig = patch_descriptor(big, cornersBig, 40)

# PLOT THE RESULT
D,match = spectral_matching2(0.5,patches1, patchesSmall, corners1, cornersSmall)
plt.figure(figsize=(12, 6))
match_euclidean = show_match2(match,corners1,cornersSmall,img1, small)

D,match = spectral_matching2(0.5,patches1, patchesBig, corners1, cornersBig)
plt.figure(figsize=(12, 6))
match_euclidean = show_match2(match,corners1,cornersBig,img1, big)
```

Add your final discussion here !

From the experiments we can assert that the choice of the patch is really important.

In the case of translation, we have seen that this type of transformation doesn't affect the correctness of the matching procedure. **But** in the case of rotation and scale the correctess of the matching procedure is affected in a negative way in fact we have seen that we get worse result respect to the translation case.

→ So I think that if we had some prior informations on what kind of change we could expect to match, we could make a more apt choiche, getting a better result!