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 [ ]:
```
%matplotlib inline
import sys
import numpy as np
import matplotlib
import numpy as np
import matplotlib.pyplot as plt
```

SIFT - a demo

The objective of this activity is to critically analyse the results you obtain by running the code and learn about the potential of SIFT descriptors for feature matching

First time with OpenCV?

So far we have used skimage. OpenCV has a wider range of Computational Vision pre-implemented functionalities.

Have a look at online tutorials

First, let us import the library

In [ ]:
```
import cv2 as cv
```

1. SIFT computation and visualization

Black & White image

If you want to know something more click on the SIFT OpenCV tutorial

We'll try out with an intensity image first

In [ ]:
```
img = cv.imread('images/Rubik1.pgm')
gray= cv.cvtColor(img,cv.COLOR_BGR2GRAY)
img_sift = None

sift = cv.SIFT_create()
kp = sift.detect(gray,None)
img_sift = cv.drawKeypoints(gray,kp,img_sift, flags=cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS) #try and delete
cv.imwrite('sift_keypoints.jpg', img_sift)
```

In [ ]:
```
fig = plt.figure(figsize=(50, 100))
fig.add_subplot(1,2,1)
#plt.imshow(cv.cvtColor(gray, cv.COLOR_BGR2RGB))
plt.imshow(img)
fig.add_subplot(1,2,2)
#plt.imshow(cv.cvtColor(img, cv.COLOR_BGR2RGB))
plt.imshow(img_sift)
```

Can you comment the output you just obtain? Do the detected features make sense?

The provided code implements the SIFT (Scale-Invariant Feature Transform) algorithm which with the keypoints are detected in a grayscale image.

OpenCV provides the function that we used *cv.drawKeyPoints()* which draws the circles on the locations of keypoints which correspond to interesting points in the image that are distinctive and can be reliably detected across different scales and orientations. If we pass the flag *cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS* to it, it will draw a circle with size of keypoint and it will even show its orientation.

In output, in fact, we get the original image where all the keypoints that have been found, the detected features seem to have sense. Most of them are located in the dotted background of the Rubik's cube where there are a lot of intensity changes and where there are regions of high contrast. Moreover inside each circle there is a line that indicates the orientation of the variation.

Color image

We try now with a color image. Notice that OpenCV uses BGR as a default color space, while the visualization functions don't (thus we need to convert BGR2RGB)

In [ ]:
```
# ANOTHER EXAMPLE
img = cv.imread('imageset/BackgroundChange/09601.jpg')
gray= cv.cvtColor(img,cv.COLOR_BGR2GRAY)
img_sift = None

sift = cv.SIFT_create()
kp = sift.detect(gray,None)
img_sift=cv.drawKeypoints(img,kp,img_sift, flags=cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS) #try and delete the

fig = plt.figure(figsize=(50, 100))
fig.add_subplot(1,2,1)
plt.imshow(cv.cvtColor(img, cv.COLOR_BGR2RGB))
#plt.imshow(img)
fig.add_subplot(1,2,2)
plt.imshow(cv.cvtColor(img_sift, cv.COLOR_BGR2RGB))
#plt.imshow(img_sift)
```

Can you comment the output you just obtain? Do the detected features make sense?

As in the b&w image, also in this case I think that the features have some sense: they are located in the points of the scene where we have a bigger number of objects / elements / changes / variations in the image. The white background has a small number of features and all of them close to the points where "something happen", in the uniform tint pieces of the scene no keypoints are detected. The cereal box is full of colored text elements and pictorial elements so here we have most of the keypoints detedted.

2. Image matching with SIFT features

We now compute matches between SIFT features from an image pair. In the example we notice a background, illumination and scale change

2.1 Detection first

In [ ]:
```
# Read and plot relevant images
img1 = cv.imread('imageset/BackgroundChange/11201.jpg')
img2 = cv.imread('imageset/BackgroundChange/18301.jpg')

# Parallel display of images
fig = plt.figure()
fig.add_subplot(1,2,1)
plt.imshow(cv.cvtColor(img1, cv.COLOR_BGR2RGB))
fig.add_subplot(1,2,2)
plt.imshow(cv.cvtColor(img2, cv.COLOR_BGR2RGB))
```

In [ ]:
```
sift = cv.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
img1_with_kp = None
img2_with_kp = None
img1_with_kp = cv.drawKeypoints(img1, kp1, img1_with_kp)
img2_with_kp = cv.drawKeypoints(img2, kp2, img2_with_kp)
```

In [ ]:
```
# Converting image1 from BGR to RGB (OpenCV design)
img1_with_kp=cv.drawKeypoints(img1, kp1, img1_with_kp, flags=cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
img2_with_kp=cv.drawKeypoints(img2, kp2, img2_with_kp, flags=cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

plt.figure(figsize=(50, 100))
plt.subplot(1,2,1)
plt.imshow(cv.cvtColor(img1_with_kp, cv.COLOR_BGR2RGB))
plt.subplot(1,2,2)
plt.imshow(cv.cvtColor(img2_with_kp, cv.COLOR_BGR2RGB))
```

2.2 Matching

The following feature matching will follow two steps:

1. Brute force matching
2. Lowe's filtering

Brute force matching is a simple: take the featue description of each feature in one image and calculate it's distance from every feature in the other image. Lowe filtering is a thresholding technique to eliminate matches with high distances.

If you have time, you may check other parameters of BFMatcher. You may try them and see how the results change!

In [ ]:
```
# BFMatcher with default params
```

```
bf = cv.BFMatcher()
```

In [ ]:
```
# Testing the top two best matches to increase matching accuracy
matches = bf.knnMatch(des1,des2, k=2)
```

In [ ]:
```
# Store all the good matches as per Lowe's ratio test
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append([m])
```

In [ ]:
```
img3 = None
img3 = cv.drawMatchesKnn(img1,kp1,img2,kp2,good,img3,flags=2)
plt.figure(figsize=(50, 100))
plt.imshow(cv.cvtColor(img3, cv.COLOR_BGR2RGB))
```

Try the above with other images pairs from the sets provided. In particular, try image pairs from

1. Imageset/2DMovement
2. Imageset/ColorChanges
3. Imageset/Random

Observe the success and failure of the two methods against different types of variations.

cv.NORM_L1 Parameter

In [ ]:
```
bf = cv.BFMatcher(cv.NORM_L1, crossCheck=False)
matches = bf.knnMatch(des1, des2, k=2)
```

In [ ]:
```
# Store all the good matches as per Lowe's ratio test
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append([m])
```

In [ ]:
```
img3 = None
img3 = cv.drawMatchesKnn(img1,kp1,img2,kp2,good,img3,flags=2)
plt.figure(figsize=(50, 100))
plt.imshow(cv.cvtColor(img3, cv.COLOR_BGR2RGB))
```

2D Movements

Test 1

In [ ]:
```
# Read and plot relevant images
img1 = cv.imread('imageset/2DMovements/06801.jpg')
img2 = cv.imread('imageset/2DMovements/09401.jpg')

# Parallel display of images
fig = plt.figure()
fig.add_subplot(1,2,1)
plt.imshow(cv.cvtColor(img1, cv.COLOR_BGR2RGB))
fig.add_subplot(1,2,2)
plt.imshow(cv.cvtColor(img2, cv.COLOR_BGR2RGB))
```

In [ ]:
```
sift = cv.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
img1_with_kp = None
img2_with_kp = None
img1_with_kp = cv.drawKeypoints(img1, kp1, img1_with_kp)
img2_with_kp = cv.drawKeypoints(img2, kp2, img2_with_kp)

# Converting image1 from BGR to RGB (OpenCV design)
img1_with_kp=cv.drawKeypoints(img1, kp1, img1_with_kp, flags=cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
img2_with_kp=cv.drawKeypoints(img2, kp2, img2_with_kp, flags=cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

plt.figure(figsize=(50, 100))
plt.subplot(1,2,1)
plt.imshow(cv.cvtColor(img1_with_kp, cv.COLOR_BGR2RGB))
plt.subplot(1,2,2)
plt.imshow(cv.cvtColor(img2_with_kp, cv.COLOR_BGR2RGB))
```

In [ ]:
```
# BFMatcher with default params
bf = cv.BFMatcher()

# Testing the top two best matches to increase matching accuracy
```

```
matches = bf.knnMatch(des1,des2, k=2)

# Store all the good matches as per Lowe's ratio test
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append([m])

img3 = None
img3 = cv.drawMatchesKnn(img1,kp1,img2,kp2,good,img3,flags=2)
plt.figure(figsize=(50, 100))
plt.imshow(cv.cvtColor(img3, cv.COLOR_BGR2RGB))
```

ADD YOUR COMMENTS HERE

These two images are different because the cereal box is rotated with a 2D movement. The matching procedure is less precise than the case before in which the two images were more similar, despite this the matching procedure is still good even if we can see some errors.

Test 2

In [ ]:
```
# Read and plot relevant images
img1 = cv.imread('imageset/2DMovements/09401.jpg')
img2 = cv.imread('imageset/2DMovements/10201.jpg')

# Parallel display of images
fig = plt.figure()
fig.add_subplot(1,2,1)
plt.imshow(cv.cvtColor(img1, cv.COLOR_BGR2RGB))
fig.add_subplot(1,2,2)
plt.imshow(cv.cvtColor(img2, cv.COLOR_BGR2RGB))
```

In [ ]:
```
sift = cv.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
img1_with_kp = None
img2_with_kp = None
img1_with_kp = cv.drawKeypoints(img1, kp1, img1_with_kp)
img2_with_kp = cv.drawKeypoints(img2, kp2, img2_with_kp)

# Converting image1 from BGR to RGB (OpenCV design)
img1_with_kp=cv.drawKeypoints(img1, kp1, img1_with_kp, flags=cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
img2_with_kp=cv.drawKeypoints(img2, kp2, img2_with_kp, flags=cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

plt.figure(figsize=(50, 100))
plt.subplot(1,2,1)
plt.imshow(cv.cvtColor(img1_with_kp, cv.COLOR_BGR2RGB))
plt.subplot(1,2,2)
plt.imshow(cv.cvtColor(img2_with_kp, cv.COLOR_BGR2RGB))
```

In [ ]:
```
# BFMatcher with default params
bf = cv.BFMatcher()

# Testing the top two best matches to increase matching accuracy
matches = bf.knnMatch(des1,des2, k=2)

# Store all the good matches as per Lowe's ratio test
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append([m])

img3 = None
img3 = cv.drawMatchesKnn(img1,kp1,img2,kp2,good,img3,flags=2)
plt.figure(figsize=(50, 100))
plt.imshow(cv.cvtColor(img3, cv.COLOR_BGR2RGB))
```

ADD YOUR COMMENTS HERE

In this case we have a small 2D rotation of the cereal box in fact, the matching procedure is really precise and accurate. Most of the connected features seem to be correct.

3D Movements

In [ ]:
```
# Read and plot relevant images
img1 = cv.imread('imageset/3DMovements/18301.jpg')
img2 = cv.imread('imageset/3DMovements/25301.jpg')

# Parallel display of images
fig = plt.figure()
```

```
fig.add_subplot(1,2,1)
plt.imshow(cv.cvtColor(img1, cv.COLOR_BGR2RGB))
fig.add_subplot(1,2,2)
plt.imshow(cv.cvtColor(img2, cv.COLOR_BGR2RGB))
```

In [ ]:
```
sift = cv.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
img1_with_kp = None
img2_with_kp = None
img1_with_kp = cv.drawKeypoints(img1, kp1, img1_with_kp)
img2_with_kp = cv.drawKeypoints(img2, kp2, img2_with_kp)

# Converting image1 from BGR to RGB (OpenCV design)
img1_with_kp=cv.drawKeypoints(img1, kp1, img1_with_kp, flags=cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
img2_with_kp=cv.drawKeypoints(img2, kp2, img2_with_kp, flags=cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

plt.figure(figsize=(50, 100))
plt.subplot(1,2,1)
plt.imshow(cv.cvtColor(img1_with_kp, cv.COLOR_BGR2RGB))
plt.subplot(1,2,2)
plt.imshow(cv.cvtColor(img2_with_kp, cv.COLOR_BGR2RGB))
```

In [ ]:
```
# BFMatcher with default params
bf = cv.BFMatcher()

# Testing the top two best matches to increase matching accuracy
matches = bf.knnMatch(des1,des2, k=2)

# Store all the good matches as per Lowe's ratio test
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append([m])

img3 = None
img3 = cv.drawMatchesKnn(img1,kp1,img2,kp2,good,img3,flags=2)
plt.figure(figsize=(50, 100))
plt.imshow(cv.cvtColor(img3, cv.COLOR_BGR2RGB))
```

ADD YOUR COMMENTS HERE

In this case we have a 3D rotation of the cereal box which has a really different position, the overlapped portion of the scene in the two images are fewer. The algorithm in this case is able to find only a few matches.

Background change

In [ ]:
```
# Read and plot relevant images
img1 = cv.imread('imageset/BackgroundChange/07701.jpg')
img2 = cv.imread('imageset/BackgroundChange/11201.jpg')

# Parallel display of images
fig = plt.figure()
fig.add_subplot(1,2,1)
plt.imshow(cv.cvtColor(img1, cv.COLOR_BGR2RGB))
fig.add_subplot(1,2,2)
plt.imshow(cv.cvtColor(img2, cv.COLOR_BGR2RGB))
```

In [ ]:
```
sift = cv.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
img1_with_kp = None
img2_with_kp = None
img1_with_kp = cv.drawKeypoints(img1, kp1, img1_with_kp)
img2_with_kp = cv.drawKeypoints(img2, kp2, img2_with_kp)

# Converting image1 from BGR to RGB (OpenCV design)
img1_with_kp=cv.drawKeypoints(img1, kp1, img1_with_kp, flags=cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
img2_with_kp=cv.drawKeypoints(img2, kp2, img2_with_kp, flags=cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

plt.figure(figsize=(50, 100))
plt.subplot(1,2,1)
plt.imshow(cv.cvtColor(img1_with_kp, cv.COLOR_BGR2RGB))
plt.subplot(1,2,2)
plt.imshow(cv.cvtColor(img2_with_kp, cv.COLOR_BGR2RGB))
```

In [ ]:
```
# BFMatcher with default params
bf = cv.BFMatcher()

# Testing the top two best matches to increase matching accuracy
```

```
matches = bf.knnMatch(des1,des2, k=2)

# Store all the good matches as per Lowe's ratio test
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append([m])

img3 = None
img3 = cv.drawMatchesKnn(img1,kp1,img2,kp2,good,img3,flags=2)
plt.figure(figsize=(50, 100))
plt.imshow(cv.cvtColor(img3, cv.COLOR_BGR2RGB))
```

ADD YOUR COMMENTS HERE

If the background of two images changes between feature detection and matching, it can affect the accuracy of the matching results.

In general we know that feature matching relies on identifying distinctive and invariant features in the images, such as corners, edges, or blobs, that can be matched across different viewpoints or under different lighting conditions. These features are often located in the foreground of the images, where the objects of interest are, and are less affected by changes in the background.

However, if the background changes significantly, it may introduce new features or remove existing ones, making it more difficult to find correspondences between the features detected in the two images.

In this case, the beackgrounds of the two images are really different and have a different illumination but the result is still quite good since the ceral box is clearly visible.

Occlusion

Test 1

In [ ]:
```
# Read and plot relevant images
img1 = cv.imread('imageset/Occlusion/30401.jpg')
img2 = cv.imread('imageset/Occlusion/31101.jpg')

# Parallel display of images
fig = plt.figure()
fig.add_subplot(1,2,1)
plt.imshow(cv.cvtColor(img1, cv.COLOR_BGR2RGB))
fig.add_subplot(1,2,2)
plt.imshow(cv.cvtColor(img2, cv.COLOR_BGR2RGB))
```

In [ ]:
```
sift = cv.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
img1_with_kp = None
img2_with_kp = None
img1_with_kp = cv.drawKeypoints(img1, kp1, img1_with_kp)
img2_with_kp = cv.drawKeypoints(img2, kp2, img2_with_kp)

# Converting image1 from BGR to RGB (OpenCV design)
img1_with_kp=cv.drawKeypoints(img1, kp1, img1_with_kp, flags=cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
img2_with_kp=cv.drawKeypoints(img2, kp2, img2_with_kp, flags=cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

plt.figure(figsize=(50, 100))
plt.subplot(1,2,1)
plt.imshow(cv.cvtColor(img1_with_kp, cv.COLOR_BGR2RGB))
plt.subplot(1,2,2)
plt.imshow(cv.cvtColor(img2_with_kp, cv.COLOR_BGR2RGB))
```

In [ ]:
```
# BFMatcher with default params
bf = cv.BFMatcher()

# Testing the top two best matches to increase matching accuracy
matches = bf.knnMatch(des1,des2, k=2)

# Store all the good matches as per Lowe's ratio test
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append([m])

img3 = None
img3 = cv.drawMatchesKnn(img1,kp1,img2,kp2,good,img3,flags=2)
plt.figure(figsize=(50, 100))
plt.imshow(cv.cvtColor(img3, cv.COLOR_BGR2RGB))
```

ADD YOUR COMMENTS HERE

In this case the occluded portion of the second image is not soo big, thus the visibility of the ceral box is not soo nicked and the most significant part of the image, where we have the most of the features is still quite visibile. For all these reasons the feature matching result is still good as much as possible.

Test 2

In [ ]:
```
# Read and plot relevant images
img1 = cv.imread('imageset/Occlusion/33501.jpg')
img2 = cv.imread('imageset/Occlusion/48901.jpg')

# Parallel display of images
fig = plt.figure()
fig.add_subplot(1,2,1)
plt.imshow(cv.cvtColor(img1, cv.COLOR_BGR2RGB))
fig.add_subplot(1,2,2)
plt.imshow(cv.cvtColor(img2, cv.COLOR_BGR2RGB))
```

In [ ]:
```
sift = cv.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
img1_with_kp = None
img2_with_kp = None
img1_with_kp = cv.drawKeypoints(img1, kp1, img1_with_kp)
img2_with_kp = cv.drawKeypoints(img2, kp2, img2_with_kp)

# Converting image1 from BGR to RGB (OpenCV design)
img1_with_kp=cv.drawKeypoints(img1, kp1, img1_with_kp, flags=cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
img2_with_kp=cv.drawKeypoints(img2, kp2, img2_with_kp, flags=cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

plt.figure(figsize=(50, 100))
plt.subplot(1,2,1)
plt.imshow(cv.cvtColor(img1_with_kp, cv.COLOR_BGR2RGB))
plt.subplot(1,2,2)
plt.imshow(cv.cvtColor(img2_with_kp, cv.COLOR_BGR2RGB))
```

In [ ]:
```
# BFMatcher with default params
bf = cv.BFMatcher()

# Testing the top two best matches to increase matching accuracy
matches = bf.knnMatch(des1,des2, k=2)

# Store all the good matches as per Lowe's ratio test
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append([m])

img3 = None
img3 = cv.drawMatchesKnn(img1,kp1,img2,kp2,good,img3,flags=2)
plt.figure(figsize=(50, 100))
plt.imshow(cv.cvtColor(img3, cv.COLOR_BGR2RGB))
```

ADD YOUR COMMENTS HERE

In this case the occluded portion of the second image is bigger. The second image is visible only patrially. The visibility of the image is compromised and this produces a worse result. Some of the matching features are wrong, in particular those in the bottom of the writing on the box which is not visible in the second image.