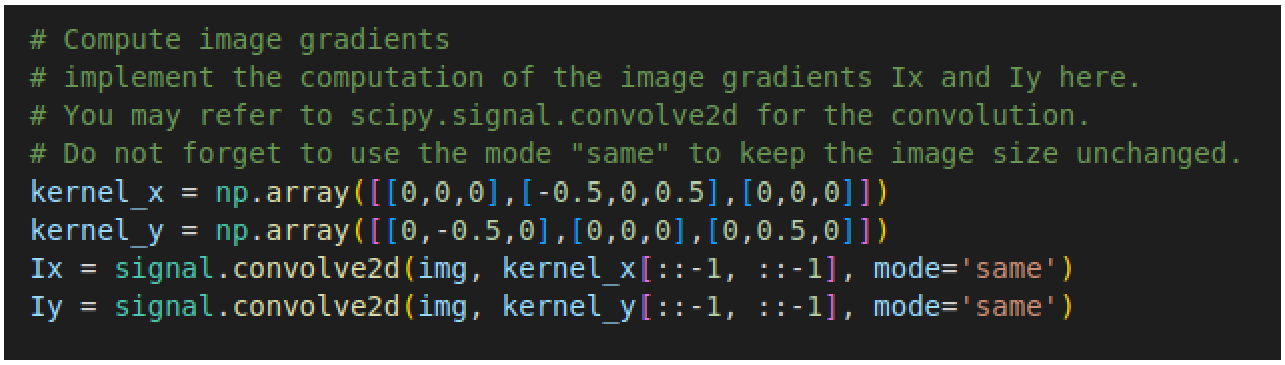
Assignment: Local features

Entiol Liko

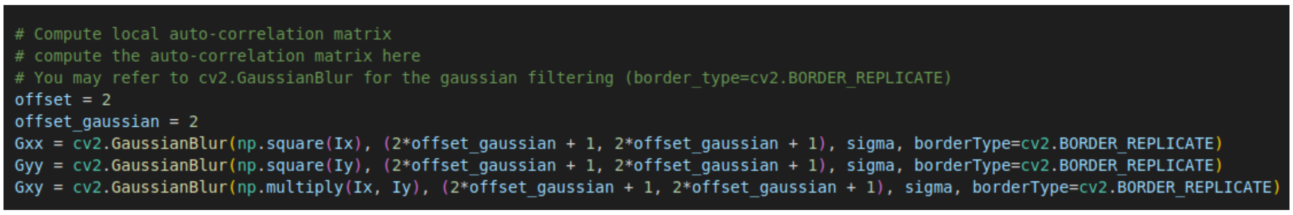
1. Harris corner detector

In the first part of the assignment we are asked to implement a Harris corner detector in order to find the points of major interest in an image. We start by computing the gradients of the image along the x and y directions.



```
# Compute image gradients
# implement the computation of the image gradients Ix and Iy here.
# You may refer to scipy.signal.convolve2d for the convolution.
# Do not forget to use the mode "same" to keep the image size unchanged.
kernel_x = np.array([[0,0,0],[-0.5,0,0.5],[0,0,0]])
kernel_y = np.array([[0,-0.5,0],[0,0,0],[0,0.5,0]])
Ix = signal.convolve2d(img, kernel_x[::-1, ::-1], mode='same')
Iy = signal.convolve2d(img, kernel_y[::-1, ::-1], mode='same')
```

The gradients of the image were computing using the convolve function of the scipy library. The above code shows the implementation and the kernel used for the calculation of the gradients.



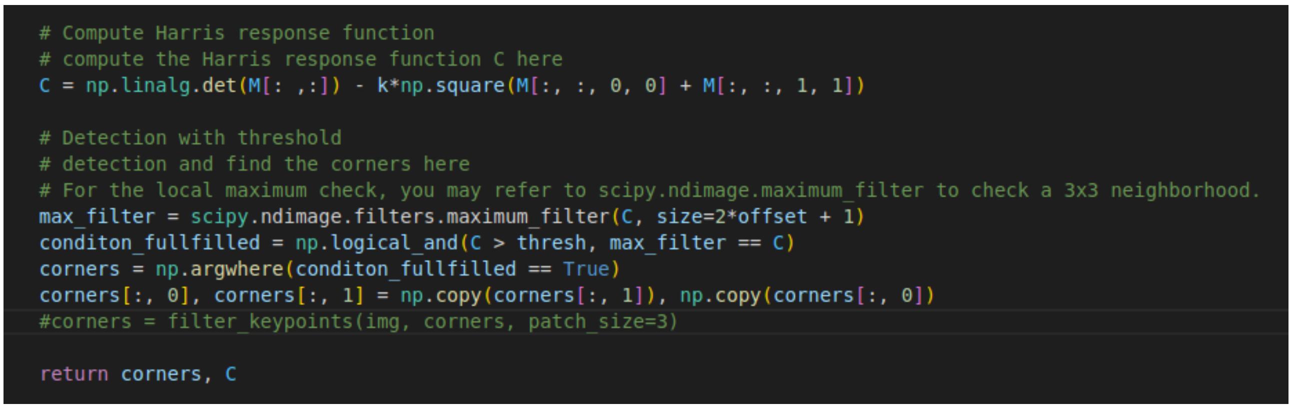
```
# Compute local auto-correlation matrix
# compute the auto-correlation matrix here
# You may refer to cv2.GaussianBlur for the gaussian filtering (border_type=cv2.BORDER_REPLICATE)
offset = 2
offset_gaussian = 2
Gxx = cv2.GaussianBlur(np.square(Ix), (2*offset_gaussian + 1, 2*offset_gaussian + 1), sigma, borderType=cv2.BORDER_REPLICATE)
Gyy = cv2.GaussianBlur(np.square(Iy), (2*offset_gaussian + 1, 2*offset_gaussian + 1), sigma, borderType=cv2.BORDER_REPLICATE)
Gxy = cv2.GaussianBlur(np.multiply(Ix, Iy), (2*offset_gaussian + 1, 2*offset_gaussian + 1), sigma, borderType=cv2.BORDER_REPLICATE)
```

Once the gradients were obtained, the next step is to compute the auto-correlation matrix. In order to make our results to be rotationally invariant, Gaussian filters were used as local weighting w of the matrix. The above code shows the calculation of the matrix M.

We now compute the Harris response function $C(i,j)$. The response function must be computed for all pixels and it has the following form:

$$C(i,j) = \det\left(M_{i,j}\right) - k\,Tr^2\left(M_{i,j}\right)$$

The very last step is to obtain the Harris corner detector.Every ‘corner’ needs to verify two conditions: $C(i,j)$ is above a certain threshold (hyperparameter) and that it is a local maxima in its 3x3 neighbourhood.

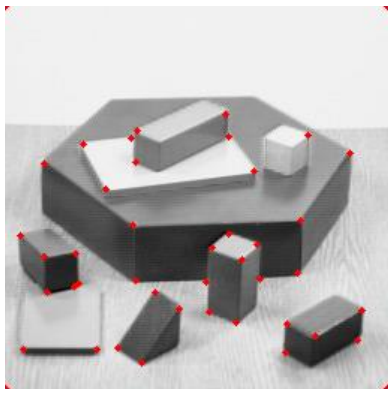


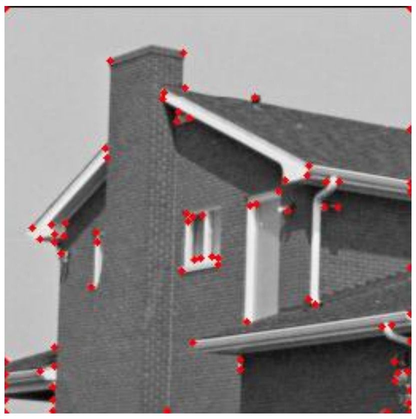
```
# Compute Harris response function
# compute the Harris response function C here
C = np.linalg.det(M[: ,:]) - k*np.square(M[:, :, 0, 0] + M[:, :, 1, 1])

# Detection with threshold
# detection and find the corners here
# For the local maximum check, you may refer to scipy.ndimage.maximum_filter to check a 3x3 neighborhood.
max_filter = scipy.ndimage.filters.maximum_filter(C, size=2*offset + 1)
conditon_fullfilled = np.logical_and(C > thresh, max_filter == C)
corners = np.argwhere(conditon_fullfilled == True)
corners[:, 0], corners[:, 1] = np.copy(corners[:, 1]), np.copy(corners[:, 0])
#corners = filter_keypoints(img, corners, patch_size=3)

return corners, C
```

The maximun_filter function was used to find the local maxima in a neighbourhood of 3 or 5 pixels both cases where tested to see which gave the best result.

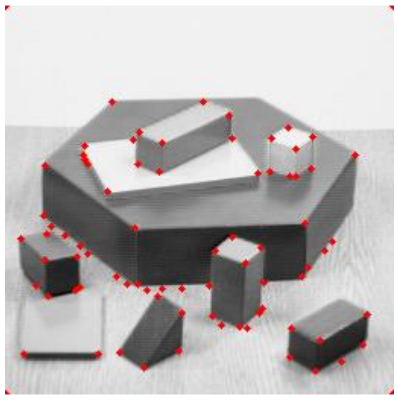
The following are some results gotten using different parameter values.

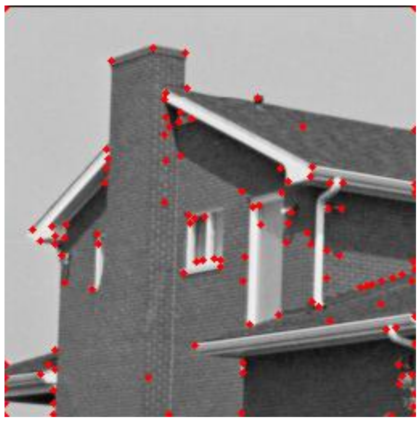
HARRIS_SIGMA = 2.0   HARRIS_K = 0.05   HARRIS_THRESH = 1e-5



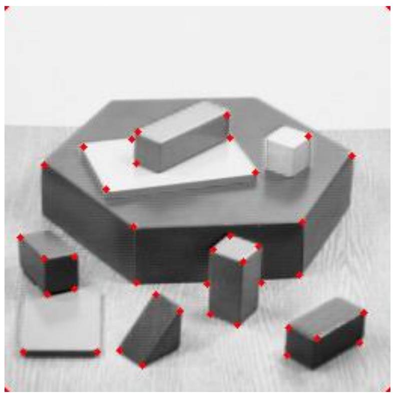


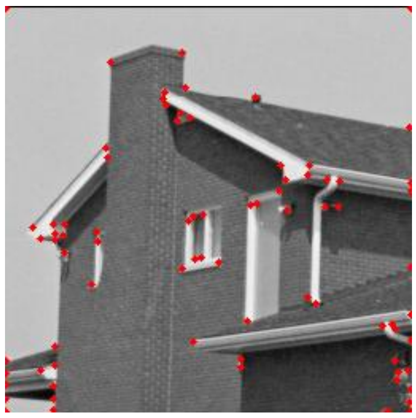
HARRIS_SIGMA = 2.0   HARRIS_K = 0.05   HARRIS_THRESH = 1e-6



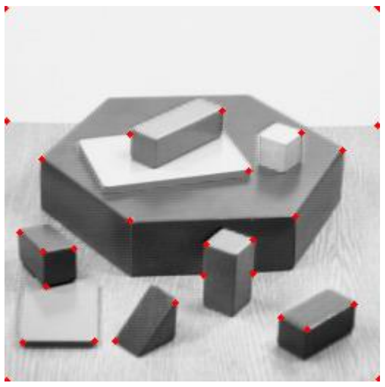


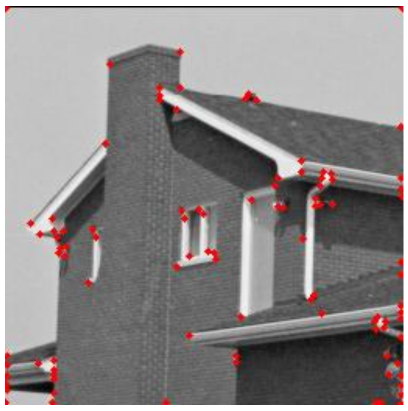
HARRIS_SIGMA = 3.0 HARRIS_K = 0.05 HARRIS_THRESH = 1e-5





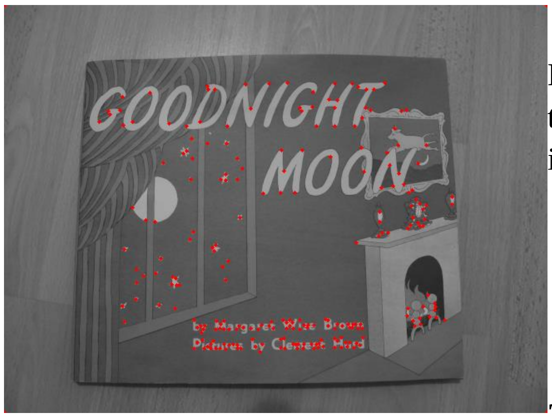
HARRIS_SIGMA = 0.5 HARRIS_K = 0.05 HARRIS_THRESH = 1e-5





2. Description & Matching

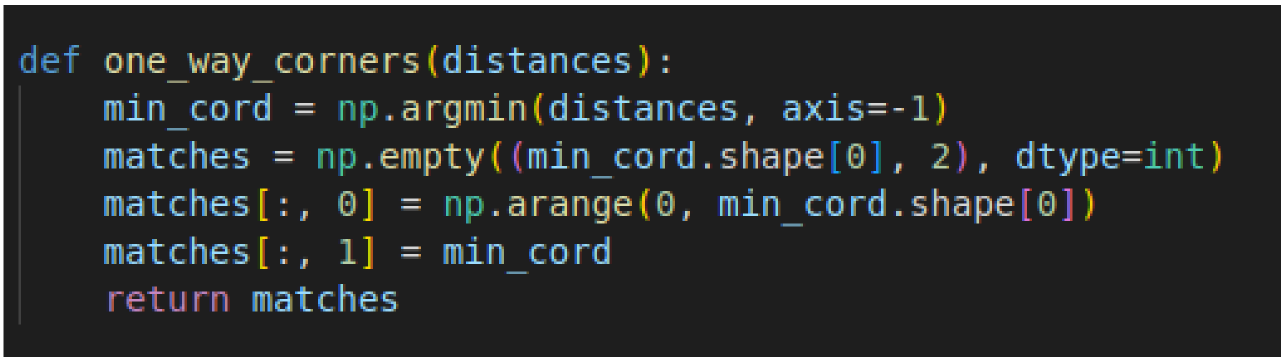
The aim of the second part of the assignment is to match features among different images. To achieve such a result, the first step is to extract local descriptors out of each previously detected keypoint. This is done by means of the *extractPatches* function, which extracts 9x9 patches around each detected keypoint. However, the keypoints must be first filtered so that we do not consider those too close to the edges.



The image on the left shows the points detected by the Harris corner detector for the image which we are going to use for the matching. I used the first parameters used in the section above.

2.1. SSD one-way nearest neighbours matching

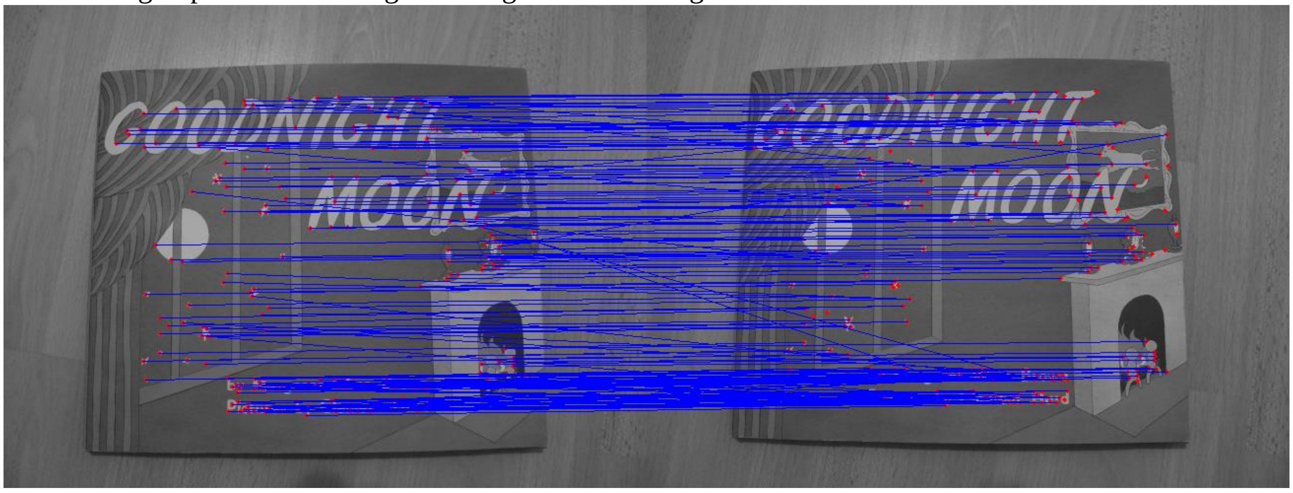
To compute the SSD one-way nearest neighbours matching the first thing to do is to compute the SSD distance between the descriptors of the features of the first and the second image.



```python
def one_way_corners(distances):
    min_cord = np.argmin(distances, axis=-1)
    matches = np.empty((min_cord.shape[0], 2), dtype=int)
    matches[:, 0] = np.arange(0, min_cord.shape[0])
    matches[:, 1] = min_cord
    return matches
```

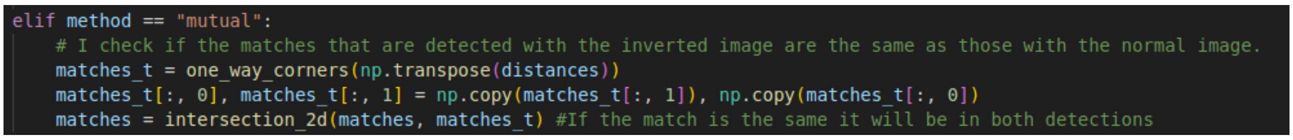
Once the distance between the descriptors is obtained, in order to perform the one-way nearest neighbours matching, for each feature descriptor of the first image we have to select the feature descriptor of the second image which has the smallest distance.

After having implemented the algorithm I got the following result:

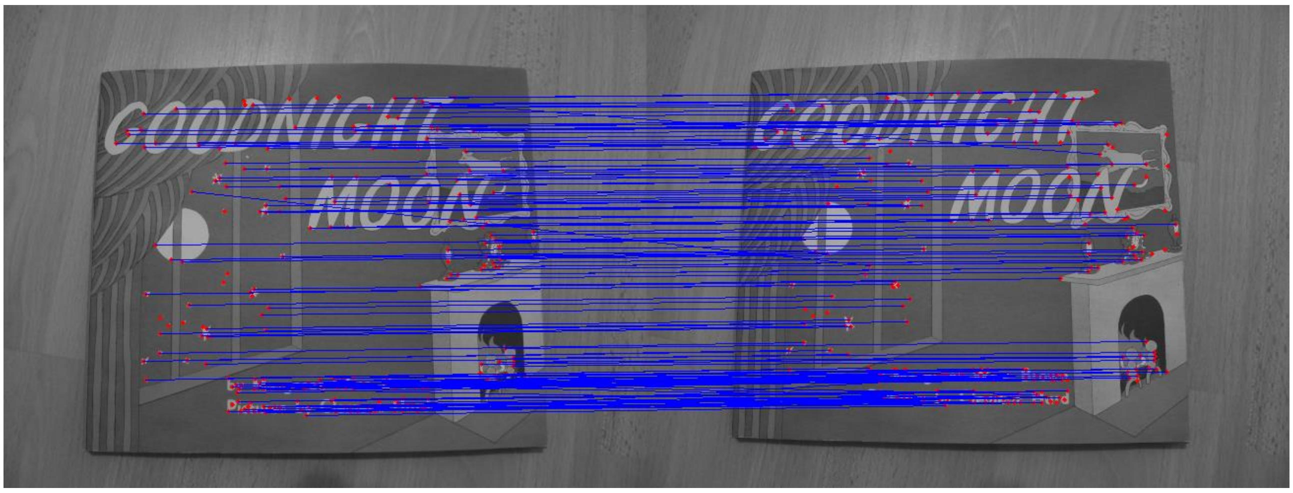


2.2. Mutual nearest neighbours matching

For the implementation of the mutual nearest neighbours matching, instead, we check that for each feature matching obtained from one-way NN, the same matching is obtained also when swapping the images. This case connected less points then the one-way method and the incorrect connection are drastically reduced.

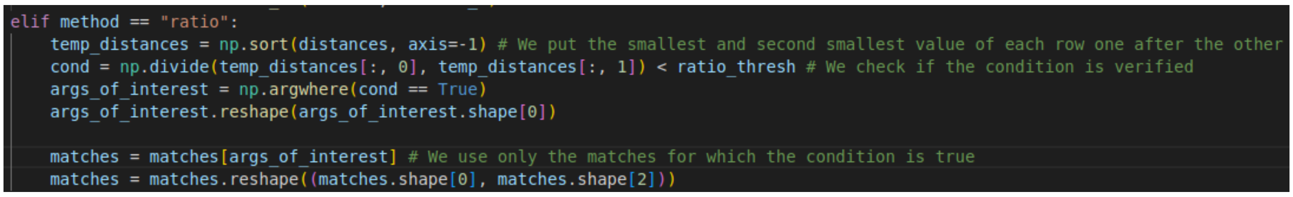


```python
elif method == "mutual":
    # I check if the matches that are detected with the inverted image are the same as those with the normal image.
    matches_t = one_way_corners(np.transpose(distances))
    matches_t[:, 0], matches_t[:, 1] = np.copy(matches_t[:, 1]), np.copy(matches_t[:, 0])
    matches = intersection_2d(matches, matches_t) #If the match is the same it will be in both detections
```

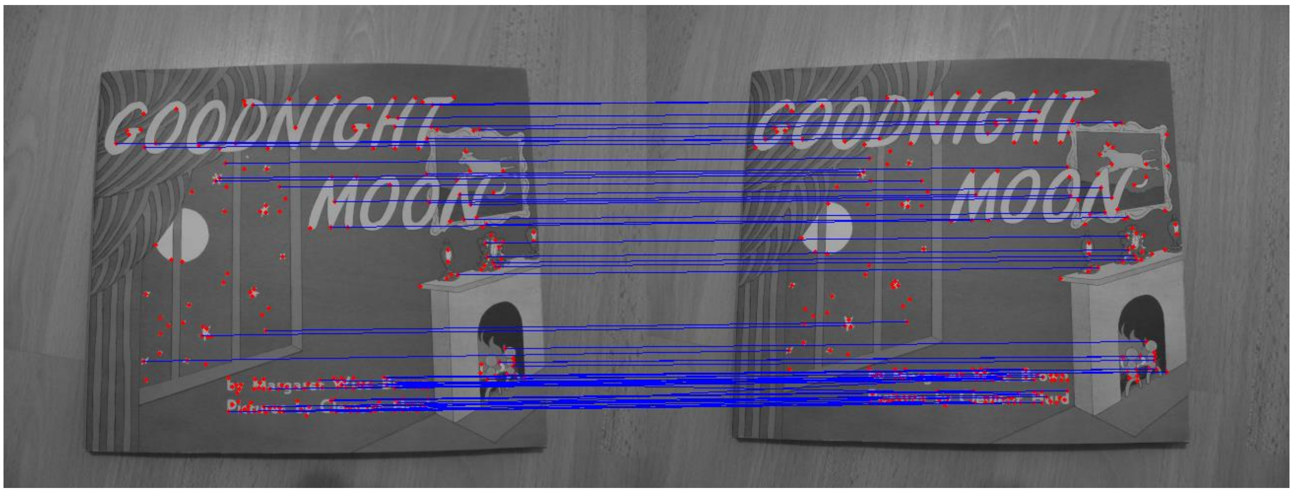


2.3. Ratio test matching

To implement Ratio test matching, instead, for each feature descriptor of the first image, I have to select the first and the second nearest neighbours (such as the two descriptors that have shortest distance from the specific feature) and check whether their ratio is below a certain chosen threshold.



```python
elif method == "ratio":
    temp_distances = np.sort(distances, axis=-1) # We put the smallest and second smallest value of each row one after the other
    cond = np.divide(temp_distances[:, 0], temp_distances[:, 1]) < ratio_thresh # We check if the condition is verified
    args_of_interest = np.argwhere(cond == True)
    args_of_interest.reshape(args_of_interest.shape[0])

    matches = matches[args_of_interest] # We use only the matches for which the condition is true
    matches = matches.reshape((matches.shape[0], matches.shape[2]))
```



Overall, the ratio test method and mutual nearest neighbours show similar result, although the MNN detects more connections between the points. From the images the connections seem to be corrent in both cases, the one-way NN shows a lot of inccorent connections.